DESARROLLO DE UNA HERRAMIENTA DE ARQUITECTURA ABIERTA PARA LA VISUALIZACION Y ANALISIS DE SEÑALES EEG

Ramiro Arango
John Jairo Naranjo

UNIVERSIDAD TECNOLOGICA DE PEREIRA

FACULTAD DE CIENCIAS BÁSICAS

MAESTRÍA EN INSTRUMENTACIÓN FÍSICA

PEREIRA

2010

DESARROLLO DE UNA HERRAMIENTA DE ARQUITECTURA ABIERTA PARA LA VISUALIZACION Y ANALISIS DE SEÑALES EEG

Ramiro Arango
John Jairo Naranjo

Trabajo de grado para optar al título de

Magister en Instrumentación Física

Director:

M. Sc. EDISON DUQUE

UNIVERSIDAD TECNOLOGICA DE PEREIRA

FACULTAD DE CIENCIAS BÁSICAS

MAESTRÍA EN INSTRUMENTACIÓN FÍSICA

PEREIRA

2010

Nota de aceptación:

_____

_____

_____

_____

Firma del Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

Pereira, _____

AGRADECIMIENTOS

A nuestras familias por su paciencia y su apoyo.

CONTENIDO

INDICE DE FIGURAS

ABREVIATURAS

TÉRMINOS BIOMÉDICOS

BCI *Brain Computer Interface.* Interfaces cerebro-computadora.

ECG *ElectroCardiography.* ElectroCardioGrafía (también aparece como EKG).

EEG *ElectroEncephalograpy.* ElectroEncefalografía.

EMG *ElectroMiography.* ElectroMiografía.

ERPs *Event Related Potential.* Potenciales endógenos relacionados con eventos.

IRDA *Intermittent Rhythmic Delta Activity.* Actividad Delta Rítmica Intermitente.

LDA *Localized-Delta Activity*. Actividad Delta Localizada.

MEG *MagnetoEncephaloGraphy*. MagnetoEncefalografía.

MRI *Magnetic Resonance Imaging*. Imagen de resonancia magnética.

PDA *Polymorphic Delta Activity.* Actividad Delta Polimórfica

PET *Positron Emission Tomography.* Tomografía de Emisión de Positrones.

REM *Rapid Eye Movement.* Movimiento rápido de los ojos.

SSW *Spikes and Sharp Waves.* Espigas y ondas agudas.

TÉRMINOS INFORMÁTICOS

ADC *Analog-Digital Conversor.* Conversor Análogo Digital.

API *Application Programming Interface*. Interface de programación de aplicación.

ASCII *American Standard Code for Information Interchange.* Código Americano Estándar para el Intercambio de Información.

ATSM *American Section of the International Association for Testing Materials.* Sección de Sociedad Internacional Americana para prueba de Materiales.

DLL *Dynamic Link Library.* Librería de Enlace Dinámico.

DSP *Digital Signal Processor*. Procesador digital de señales.

FFT *Fast Fourier Transform.* Transformada rápida de Fourier.

GNU *General Public License.* Licencia Pública General.

ICA *Independent Component Analysis.* Análisis de componentes independientes.

MMX *Matrix Math eXtension.* Conjunto de instrucciones Multimedia.

RAM *Random Access Memory.* Memoria de acceso aleatoria.

VCL *Visual Component Library.* Librería de Componentes Visuales de Borland.

XML *EXtensible Markup Language.* Lenguaje de marcas expandible.

INTRODUCCIÓN

El sistema nervioso funciona mediante contactos eléctricos entre neuronas, configurando circuitos específicos para cada función neurológica. La electroencefalografía es una técnica exploratoria no invasiva, que permite registrar la actividad bioeléctrica (ondas cerebrales) de las neuronas de la corteza cerebral, mediante unos aparatos adecuados y la colocación previa de unos electrodos, en unas posiciones estándar. El registro gráfico obtenido se denomina electroencefalograma (EEG), y consiste en una sucesión de ondas de diferentes frecuencias y amplitudes.

Las llamadas "ondas cerebrales" son en realidad curvas formadas por el voltaje o diferencia de potencial entre 2 electrodos en función del tiempo. Los registros de EEG sirven para buscar la base orgánica y entender el funcionamiento eléctrico de las afecciones. Se emplea en las epilepsias, los trastornos de la memoria, la demencia, los trastornos de aprendizaje y los cuadros psiquiátricos. Gracias al avance de la tecnología, dichos registros hoy se obtienen en formato digital, permitiendo que las valoraciones de ciertos eventos se puedan llevar a cabo por el especialista de manera más acertada. En los grupos de investigación de las universidades del eje cafetero se han presentado varios proyectos que, utilizando diversas técnicas, logran extraer características e inclusive clasificar señales de EEG y ECG, pero hasta ahora pocos de ellos han trascendido a la labor médica pues solo se llega a cumplir con los objetivos académicos.

Para el desarrollo de este trabajo, nos propusimos los siguientes objetivos:

Objetivo general:

Desarrollar una herramienta que visualice las señales EEG de la forma usual que utilizan los programas de análisis de EEG, con un sistema abierto que permita incorporar nuevas rutinas de análisis con módulos independientes (desarrolladas ya bien sea en Matlab o en un lenguaje compilado como Lenguaje C), que pueden adicionársele o quitársele en cualquier momento.

Objetivos específicos:

- Implementar rutinas que permitan acceder a los registros EEG de los Electroencefalógrafos comerciales.
- Codificar rutinas para visualización de señales EEG, con escala de voltaje y tiempo configurables por el usuario.
- Investigar sobre los diferentes métodos de interpolación en 2D, e implementar algunos para generar visualización topográfica de los registros o análisis de EEG.
- Utilizar una decodificación de video apropiada a la forma en que se graban las sesiones de EEG, y crear un método de sincronización con la visualización de las señales.
- Definir un estándar para comunicarse con rutinas desarrolladas en Matlab®, para el tratamiento de señales EEG y algunas rutinas de ejemplo que lo utilicen.
- Definir un estándar para comunicarse con módulos DLLs para el tratamiento de señales EEG, y algunas rutinas de ejemplo que lo utilicen.
- Implementar un sistema de configuración para adicionar o eliminar los módulos desarrollados.

Para cumplir con los objetivos de este proyecto, en primera instancia se evaluaron los diferentes programas que se usan para el análisis de registros EEG y de esta manera, conocer la forma usual de mostrar los registros para los especialistas (se pretendió hacer una herramienta con la cual el especialista esté familiarizado con la forma de presentar los resultados), luego se analizó la documentación sobre los diferentes formatos utilizados para guardar los registros de EEG, así como las librerías de dominio público que podrían simplificar la labor de programación. Con base en esta información se procedió a codificar las rutinas de acceso a archivos y visualización de los mismos.

Se implementó la visualización topográfica y el acceso al video que acompaña a los exámenes de EEG. Luego se procedió a definir un estándar para los módulos a incorporarse y la forma de comunicarse con el programa (paso de parámetros, retorno de valores, etc.), bien sea para módulos implementados en un lenguaje compilado o un lenguaje interpretado. Se hicieron pruebas básicas con algunos módulos que implementen algoritmos simples, y en el momento nos encontramos en la fase de validación con algunas rutinas más elaboradas, y de aplicación en epilepsia.

Hay varios aspectos que consideramos destacables en la implementación, aparte de la arquitectura abierta ya mencionada, que permiten un abanico de posibilidades y futuros proyectos, utilizando como base este trabajo.

El primero de ellos es la implementación de la visualización topográfica, de un gran valor tanto académico, como clínico, al permitir ver de forma más directa las zonas del cerebro donde se presenta actividad eléctrica. Esta característica permitiría, entre otros: la ubicación precisa de los focos epilépticos; la correlación entre los estímulos y las señales, en investigaciones de potenciales evocados; y, porque no, en investigaciones de Interfaces Cerebro Computador.

Otro de los puntos que consideramos destacables en este proyecto: la inclusión de rutinas de acceso y visualización de video que acompaña a los exámenes de EEG, lo que permite al especialista hacer un diagnóstico más acertado, al poder correlacionarlo con el registro de las señales EEG. Gracias a que permite visualizar los eventos cuantas veces se quiera, en completa sincronización con los registros, serviría como asistencia en la detección manual de artificios debidos a movimientos del paciente. Esta característica del programa también se podría aprovechar, utilizándolo como herramienta muy valiosa en el entrenamiento de estudiantes en la interpretación de los registros.

Conviene también mencionar la gran cantidad de siglas, y la disparidad de algunas de ellas, en el contexto en que desenvuelve el proyecto. Mientras que en algunas partes encontramos las siglas provenientes de palabras en español, en otras las encontramos provenientes de palabras en inglés, y nos enfrentamos al dilema de cuál de ellas utilizar: nuestro idioma nativo es español, pero el idioma nativo de los términos científicos es el inglés, ya que la mayor parte de las investigaciones y de las publicaciones se han hecho en aquel idioma. Finalmente, optamos por dejarlas en inglés, para hacerlas concordar con la amplia literatura que se encuentra en el medio, e incluir una pequeño guía de abreviaturas, para que el lector pueda fácilmente consultar su proveniencia, su significado y su homólogo en español.

1. MARCO TEÓRICO

En este capítulo se presentarán las señales EEG, su origen y la evolución de sus registros, así como un panorama de las diversas formas en que los especialistas médicos pueden utilizar los registros EEG como herramienta de diagnóstico y/o de investigación. Luego se muestran algunos aspectos técnicos que hubo que tener en cuenta a la hora de realizar la aplicación.

1.1 ORIGEN Y NATURALEZA DE LAS SEÑALES EEG

Las señales EEG son bioseñales y como tal, son variables que pueden ser medidas, monitoreadas, y que reflejan de alguna manera un estado biológico funcional. Cuando la señal medida tiene origen eléctrico, en cualquier organismo, la bioseñal se toma a partir de alguna corriente producida por el cambio de potencial eléctrico sobre algún tejido, órgano o sistema celular [1].

En particular las señales EEG se consideran generadas principalmente por la actividad de las neuronas piramidales de la capa IV de la corteza cerebral, que producen corrientes macroscópicas debido al alineamiento de las dendritas apicales (en las figuras 1 y 2 y 3 se representan modelos de una y varias neuronas piramidales de la capa IV). La amplitud de la señal EEG se relaciona con el grado de sincronía con el cual las neuronas corticales interactúan. La excitación sincronizada de un grupo de neuronas produce una señal de gran amplitud en el cuero cabelludo ya que las señales originadas en las neuronas individuales se suman de forma coherente en el tiempo. [2][3].

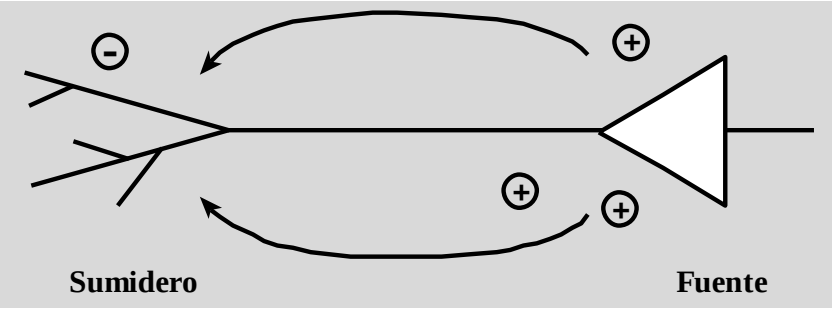


Figura 1. Ilustración de una célula piramidal, donde se muestra esquemáticamente el árbol dendrítico a la izquierda, el cuerpo celular y el axón de la derecha. Un potencial sináptico crea un sumidero de corriente en el árbol dendrítico y una fuente de corriente en el soma (Adaptado de Nunez, P.L. 1981. Electric Field of the Brain. New York: Oxford University Press).

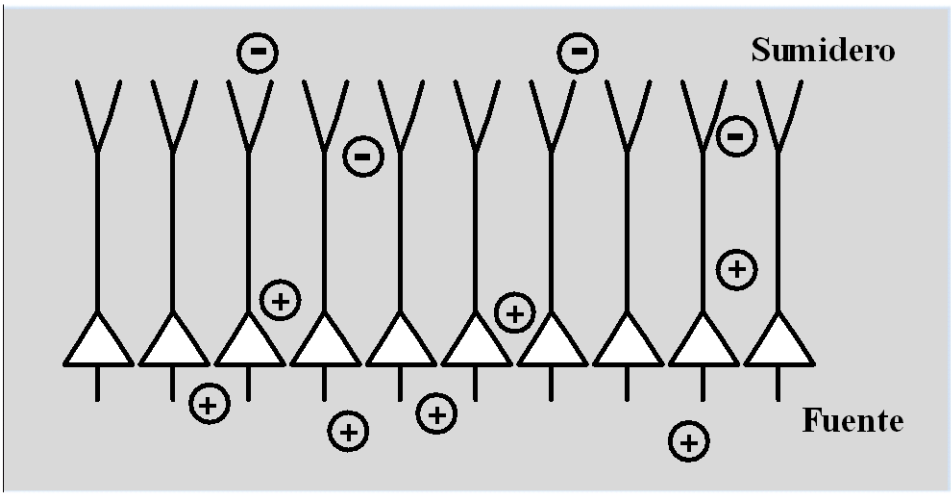


Figura 2. Alineación de las células piramidales. Estas células se alinean de una manera muy invariable, con grandes sumideros en las dendritas en una dirección y las fuentes en los somas en la otra. En este esquema se supone que las células están sincronizadas, y se crean fuertes dipolos sincronizados, ya que los flujos de las corrientes a partir de células individuales no se cancelan entre sí (Adaptado de Nunez, P.L. 1981. Electric Field of the Brain. New York: Oxford University Press).

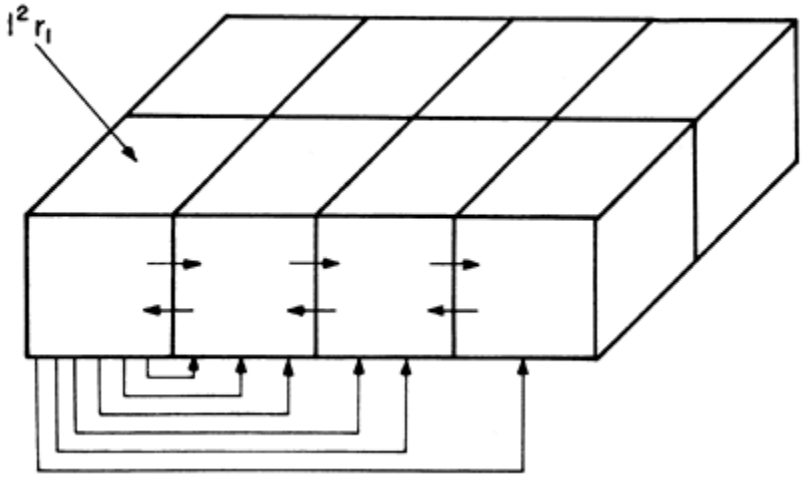


Figura 3. Modelo para la representación de una superficie cortical dividida en elementos de volumen que están interconectados por las fibras de asociación y otras intracorticales. (Adaptado de Nunez, P.L. 1981. Electric Field of the Brain. New York: Oxford University Press).

Las señales registradas a través del cuero cabelludo, en general, tienen amplitudes que oscilan entre unos pocos microvoltios hasta aproximadamente 100 μV y un contenido de frecuencia entre 0.5 a 40 Hz. En un registro de EEG normalmente se utilizan varios canales para monitorear diferentes zonas cerebrales (Figura 4). La Federación Internacional de Sociedades de Electroencefalografía y Neurofisiología Clínica recomendó el uso de un estándar en el posicionamiento de electrodos conocido como 10-20, (más adelante se detalla sobre esta convención), que consisten en 21

electrodos posicionados en sitios específicos de la corteza cerebral, sin embargo para ciertos registros se utilizan más canales para tener mejor resolución espacial.

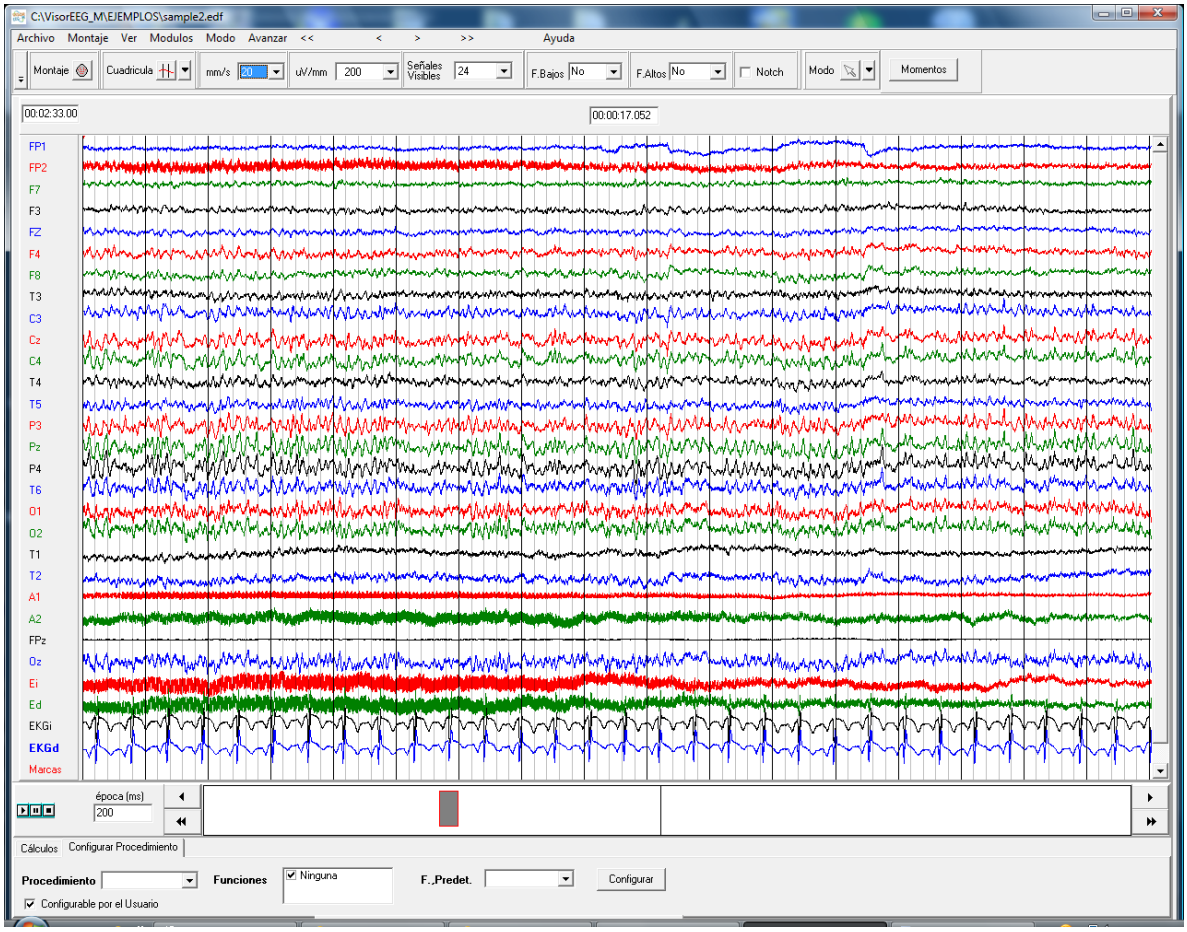


Figura 4. Registro EEG típico de un adulto (Tomado de una de las ventanas del VisorEEG_M)

1.2 EVOLUCIÓN DE LOS REGISTROS EEG

Los primeros intentos por registrar la actividad eléctrica del cerebro se deben a Richard Caton (1842- 1926). Éste fue un médico que ejerció en Liverpool, y se interesó profundamente en los fenómenos electrofisiológicos y recibió una beca de la asociación Médica Británica, para explorar los fenómenos eléctricos de los hemisferios cerebrales expuestos de conejos y monos. Según Brazier (1961), Caton presentó sus resultados a la asociación el 24 de agosto de 1875, y un informe muy breve de 20 líneas apareció posteriormente en el ***British Medical Journal***. Un informe más detallado fue presentado en la misma revista en 1877 con experimentos de más de 40 conejos, gatos y monos [4].

En sus inicios, los registros de la actividad eléctrica del cerebro utilizaban un simple galvanómetro, y fue mejorando a medida que la tecnología de dichos instrumentos avanzaba, se usaron entonces galvanómetros *d'Arsonval*, electrómetros capilares (Lippmann y Marey [12]), hasta el galvanómetro

de corriente presentado por Einthoven en 1903, un instrumento muy sensible y preciso que se utilizó como estándar por décadas.

Berger en 1929 desarrolló un sistema de registro de la actividad eléctrica cerebral, pero no fue hasta la década de los cuarenta cuando comenzó a utilizarse de modo rutinario en gran número de pacientes [5].

Ya en años más recientes se utilizan sistemas integrados que incluyen varios canales con electrodo, amplificador diferencial, filtro y registro con aguja por cada canal. Algunas mejoras en el registro Multicanal en cuanto a la visualización incluyeron la utilización de papel cuadriculado y opciones de cambio de escala, pero el gran avance se tiene cuando se logra digitalizar las señales (utilizando sistemas con ADC), y así poder integrar el registro con sistemas informáticos en los cuales se pueden almacenar los registros.

Nuevamente las mejoras en el registro y almacenamiento del EEG van de la mano de los avances en la tecnología, sobre todo de la informática, en cuanto la calidad de los registros mejora cuando se digitaliza utilizando más bits por canal, a su vez esto implica un gran volumen de datos por procesar. También fue crucial el avance en cuanto a los tipos de electrodos utilizados, tanto en sensibilidad como en tamaño. Hoy en día se tienen varias opciones de acuerdo a las necesidades: desechables (con y sin gel), electrodos de disco reusables (de oro, plata o acero inoxidable); gorros y capas de electrodos; electrodos de aguja, etc. [13]. Los sistemas informáticos modernos permiten una configuración variable de estímulos y frecuencia de muestreo, y algunas están equipadas con simples o avanzadas herramientas de procesamiento de señales para el análisis de los registros.

1.3 PROPÓSITO DE LOS REGISTROS EEG

La medición de la actividad eléctrica espontánea de la corteza cerebral mediante un EEG es de gran importancia para diagnosticar, entre otros, la presencia y tipo de trastornos convulsivos, buscar las causas de la convulsión y para evaluar lesiones en la cabeza tales como tumores, infecciones, enfermedades degenerativas y alteraciones metabólicas que afectan al cerebro, razón por la cual los registros de EEG son una herramienta vital para que los médicos especialistas puedan diagnosticar efectivamente diversas patologías. Gracias al avance de la tecnología, dichos registros hoy se obtienen en formato digital, permitiendo que las valoraciones de ciertos eventos se puedan llevar a cabo por el especialista de manera más acertada; pero también, cada vez es más frecuente la aplicación de técnicas avanzadas de procesamiento digital de señales a dichos registros, logrando ejecutar de manera automática algunas de las labores realizadas por los especialistas y así aliviar la pesada carga que le representa el gran volumen de datos a procesar [14].

Algunas de las aplicaciones de los registros EEG se listan a continuación:

(a) Detección de estado de alerta de vigilancia, coma y muerte cerebral

(b) Localización de la zona de los daños tras una lesión de cabeza, derrame cerebral, y tumores

(c) Ensayos de las vías aferentes (por potenciales evocados)

(d) Seguimiento de la participación cognitiva (Ritmo Alfa)

(e) Biorrealimentación

(f) Control de profundidad de anestesia (servo anestesia)

(g) Investigación de la epilepsia y la localización de origen de focos

(h) Pruebas de los efectos de drogas en epilepsia

(I) Asistencia para la extirpación experimental cortical de focos epilépticos

(j) Seguimiento del desarrollo del cerebro

(k) Ensayos sobre los efectos convulsivos de drogas

(l) Investigación de los trastornos del sueño y la fisiología

(m) Investigación de los trastornos mentales

(n) Proporcionar un sistema de grabación de datos híbrido junto con otras modalidades de imágenes.

1.4 DETECCIÓN DE PATOLOGÍAS CEREBRALES BASADAS EN EL REGISTRO EEG

Los médicos deben identificar los signos, síntomas y señales orgánicas que sirvan de referentes para determinar los estados de normalidad o anormalidad, asociada a posibles enfermedades.

La identificación se realiza a través de la percepción sensorial, en ocasiones con mediación de instrumentos o dispositivos elementales que magnifican las señales orgánicas o facilitan el juicio sobre el grado de normalidad o anormalidad de los diferentes estados funcionales del organismo.

Este modelo tiene varias limitaciones: alta subjetividad cuando sólo se interpreta por un médico, imposibilidad de almacenamiento y réplica para un futuro análisis en caso de requerirse una asesoría de mayor precisión en el diagnóstico o en las decisiones terapéuticas, por último, se presenta el sesgo y otras clases de error, a pesar de la destreza generada por el entrenamiento y la formación del recurso humano. De ahí que, a menudo, se requiera de equipos y ayudas de tecnología que faciliten la exploración y auscultación que de manera objetiva mejoren la calidad de las decisiones de tipo clínico; o en otros casos, que provean de la información suficiente al especialista con el objetivo de que sirva de soporte a una adecuada solución del problema. [1]. El electroencefalograma es un instrumento elemental para identificar los estados funcionales del cerebro.

La Electroencefalografía clínica correlaciona la actividad del sistema nervioso central tanto en funcionamiento normal como en disfunciones y enfermedades, con ciertos patrones del EEG con base en resultados empíricos obtenidos con anterioridad, pero cada vez con más frecuencia las investigaciones en este campo conllevan a una explicación científica que exponga los procesos

fisiológicos elementales subyacentes [15]. En gran medida estos resultados son consecuencia de las facilidades de que disponen los equipos de electroencefalografía moderna.

Algunas enfermedades son diagnosticadas con base en los patrones anormales en las señales EEG ya sea por la localización espacial (completa, local, unilateral, bilateral, etc.), o por su persistencia en el tiempo (breve e intermitente o prolongada y persistente); estos patrones anormales tienen que ver con la ausencia o persistencia de las ondas asociadas a los ritmos cerebrales ya sea en vigilia o en estados del sueño, o con respuestas funcionales anormales en estados conscientes. Inclusive, las anormalidades en las ondas cerebrales permiten determinar el grado de integridad fisiológica de la corteza cerebral en pacientes en estado de inconsciencia (estado de coma o vegetativo, o inducidos por anestesia).

A continuación se mencionarán las características básicas que se tienen en cuenta para clasificar un registro de EEG y algunas de las patologías asociadas a patrones específicos de dichas características.

1.5 RITMOS EEG

Las características primarias de un registro EEG a las cuales se les presta especial atención son la amplitud y el contenido de frecuencias de una señal. A dicho contenido se le denomina comúnmente como ritmo del EEG:

- Delta: Menores a 4 Hz
- Teta: Entre 4 a 7 Hz
- Alfa: Entre 8 a 13 Hz
- Beta: Entre 14 a 30 Hz
- Gamma: Mayores a 30 Hz

Aunque no hay una explicación científica a cada uno de los ritmos cerebrales, la correlación entre contenido en frecuencia y amplitud sí puede asociarse con la actividad en sincronía o no de las células cercanas a un electrodo. Los ritmos de baja amplitud y altas frecuencias reflejan un cerebro activo asociado con un estado de alerta o cuando se está soñando, mientras que las señales de gran amplitud y bajas frecuencias se asocian con estados de somnolencia o cuando se duerme sin la aparición de sueños. Esta relación es lógica, debido a que cuando la corteza está procesando información de forma más activa, ya sea por una entrada sensora o por algún proceso interno, el nivel de actividad de las neuronas corticales es relativamente alto, pero igualmente desincronizado [2].

En otras palabras, cada neurona, o un grupo muy pequeño de neuronas resulta envuelto vigorosamente en un aspecto ligeramente diferente de una tarea cognitiva compleja; dispara de forma rápida, pero no de forma simultánea con la mayoría de sus vecinos. Esto conduce a una baja sincronía, de forma que la amplitud del EEG es baja. En contraste, durante el sueño profundo, las neuronas corticales no están procesando información, y una gran cantidad de ellas están físicamente excitadas por una entrada rítmica común. La frecuencia, o tasa de oscilación de un ritmo EEG es

parcialmente sostenido por la actividad de entrada del tálamo. Esta parte del cerebro está constituida por neuronas que poseen propiedades de marcapasos, las cuales tienen la actividad intrínseca de generar un patrón rítmico de disparo auto sostenido. En este caso la sincronía es alta, de forma que la amplitud del EEG es grande [16].

Para una frecuencia dada, normalmente una señal de gran amplitud puede considerarse anormal (en algunos casos esto se puede confundir con artificios debidos al movimiento del paciente, de ahí la importancia de tener un registro del video simultaneo con el EEG), esto es cierto para todas las frecuencias, pero sobre todo para las altas frecuencias (ritmo beta). Las bajas amplitudes en cambio, podrían estar asociadas a una disminución peligrosa de los voltajes cerebrales, pero considerando la baja sincronización en estados de vigilia podrían tratarse de valores normales.

En los estudios de las fases de sueño, se correlacionan ciertos patrones con patologías como problemas metabólicos y/o cerebro vasculares. Por ejemplo, se asocian problemas cerebro vascular con ritmos alfa intermitentes en pacientes de edad avanzada [17]. Las similitudes entre las frecuencias de ritmo alfa y el temblor fisiológico de los dedos han sido discutidas por Isokawa y Komisaruk (1983). Las diferencias en la amplitud de las ondas Alfa, cuando se compara la actividad por hemisferios, parece que se relaciona con el hemisferio dominante, pero también existen estudios que relacionan la falta de diferencias con la depresión endógena [18].

También se utiliza el EEG para detección de los tumores cerebrales, teniendo en cuenta ciertos patrones: Actividad Delta Polimórfica (PDA) o Actividad Delta Localizada (LDA). Estas ondas lentas se localizan habitualmente en el lado del tumor, con formas de onda irregulares (polimórfico) y continuas. Actividad delta rítmica intermitente (IRDA) o actividad delta monorrítmica sinusoidal (MSDA). Perdida de actividad eléctrica en las áreas cercanas al sitio donde se encuentra el tumor. Los gliomas del lóbulo frontal tienden a causar descargas locales de alto voltaje, a veces más de 100 μV.

En algunos casos, el desarrollo de alteraciones del EEG paroxístico puede predecir la aparición de trastornos convulsivos. Pueschel (1991) y Katada (2000) demostraron una disminución progresiva de las frecuencias alfa en la región occipital en una edad más temprana en el síndrome de Down, que en otros tipos de retraso mental.

Un análisis más detallado del EEG implica tener en cuenta las formas de onda típicas. Las ondas en el EEG son reconocidas principalmente por su forma, y de manera secundaria por su frecuencia; estas incluyen ondas que pueden ser normales o anormales según su contexto (como las espigas y ondas agudas).

1.6 FORMAS DE ONDA

Las espigas y ondas agudas (*SSW –Spikes and Sharp Waves*) son formas de onda transitorias que sobresalen entre las de la actividad de fondo del EEG con un patrón temporal irregular e imprevisible (actividad paroxística). Su presencia indica un comportamiento neuronal extraño comúnmente encontrado en pacientes que sufren de crisis epilépticas [19]. Debido a su relación con

los ataques epilépticos, a las SSW se les refiere como interictales, debido a que ocurren entre eventos ictales o eventos epilépticos.

La definición clínica de las SSW es ambigua, pero ambos tipos de forma de onda generalmente se caracterizan por un arrancón inicial muy empinado. Una espiga se diferencia de una onda aguda por su duración: una espiga tiene una duración dentro del rango de 20−70 ms, mientras que una onda aguda dura de 70−200 ms. Aunque la morfología de la forma de onda es esencialmente monofásica, no es extraño observar formas de onda bi o trifásicas. La morfología de la forma de onda depende, naturalmente, de la ubicación del electrodo en el cuero cabelludo.

Formas de Onda asociadas a las fases de sueño: El cerebro tiene tres estados funcionales esenciales: vigilancia, sueño con movimiento rápido de los ojos (*REM–Rapid Eye Movement*), y sueño sin REM. Los dos estados del sueño, comúnmente conocidos como sueño REM y sueño no-REM, suceden varias veces durante una noche. El sueño no–REM es un estado de espera asociado con el descanso del cerebro y de las funciones corporales. Los ritmos EEG lentos de gran amplitud durante un sueño no–REM indican un alto grado de sincronía de las neuronas corticales subyacentes. A este estado se le puede subdividir en cuatro etapas relacionadas con el grado de profundidad del sueño. Cada una de ellas contiene formas de onda características (Complejos K, Ondas V, Ondas Lambda, POSTS, Espigas de sueño, Ondas mu, y transitorios epilépticos benignos del sueño). Para una descripción detallada de las formas de onda asociados con estas etapas consultar [1].

EEG ictal: Durante un ataque epiléptico al EEG se le refiere como EEG ictal, manifestado por un ritmo anormal con un incremento súbito de amplitud. El inicio de un ataque epiléptico también se ha asociado con un cambio súbito en el contenido de frecuencia que normalmente evoluciona en un ritmo con un patrón de ondas afiladas. El EEG ictal puede tener una gran variabilidad entre los ataques, haciendo difícil su detección tanto de forma manual como automática.

Algunos estudios correlacionan las formas de onda lentas y prolongadas en infantes prematuros con hemorragias interventriculares, principalmente sobre el área rolándica [20][21].

1.7 OTRAS APLICACIONES DE LOS REGISTROS EEG

Las funciones cerebrales de las diferentes regiones corticales históricamente se han descubierto teniendo en cuenta las lesiones cerebrales y sus manifestaciones en las funciones orgánicas y/o cambios de comportamiento, pero con el EEG también se pueden llevar a cabo investigaciones de este tipo, por ejemplo las que relacionan la respuesta cortical a estímulos sensoriales y todas aquellas que implican actividades neurocognitivas. La importancia del electroencefalograma (EEG) para investigar las funciones neurocognitivas fue reconocida ya por su descubridor, el mismo Hans Berger. En su primer escrito sobre el EEG en humanos presentó el tema como una pregunta: "¿Será posible demostrar los procesos intelectuales por medio de un electroencefalograma?" [22]. Y le dio una respuesta positiva en la misma publicación cuando describió el bloqueo-alfa durante el procesamiento cognitivo como principal objeto de correlación de los estados mentales. Así, el efecto Berger fue el punto de partida de la investigación EEG-neurocognitiva.

La electroencefalografía como método general para la investigación de las funciones del cerebro humano incluye formas de determinar la reacción del cerebro a una variedad de estímulos. El campo de investigación dedicado a la detección, cuantificación y análisis fisiológico de los cambios leves en el EEG que están relacionados con acontecimientos particulares han sido tema de creciente interés en los últimos años, la mayoría de estos tópicos se tratan con el término genérico de ERP (potenciales relacionados con eventos), o las versiones más concretas potenciales evocados (PE), y potenciales sensoriales.

Utilizando las técnicas del EEG y el MEG (*MagnetoEncephaloGraphy)* para tener una buena resolución temporal y técnicas de MRI (*Magnetic Resonance Imaging*) y PET (*Positron Emission Tomography*), para tener una buena resolución espacial abundan las investigaciones en éste campo (para una completa revisión consultar [23]). El registro EEG se utiliza por ejemplo para correlacionar las frecuencias asociadas a los registros con procesos neurofisiológicos, los potenciales endógenos relacionados con eventos (ERPs), los cambios sostenidos de los niveles DC en los potenciales corticales antes o durante el desempeño mental.

Un desarrollo relativamente reciente en la neurofisiología aplicada son las llamadas interfaces cerebro-computadora (BCI, por las siglas en inglés de *Brain Computer Interface*), por medio de los cuales se extraen automáticamente características las señales de EEG y se utilizan para operar los dispositivos controlados por computadora para ayudar a los pacientes que tienen comprometidas las funciones motoras, como es el caso de los pacientes tetrapléjicos. Este nuevo enfoque fue posible gracias a los avances en los métodos de análisis del EEG y en tecnología de la información, unida a una mejor comprensión de la correlación de funciones psicofisiológicos con ciertas características del EEG. Como objetivo a largo plazo de esta tecnología de la BCI es disponer de un sistema que permite la comunicación directa entre el cerebro y la computadora.

1.8 DIAGNÓSTICO ASISTIDO POR EL COMPUTADOR

Actualmente, los adelantos relacionados con el análisis de señales y la automatización de sistemas, hacen que el modelo de diagnóstico presente cada vez mayor interés en el empleo de los sistemas automáticos de procesamiento e identificación de estados funcionales del organismo, los cuales brindan soporte al personal médico especialista que interpreta y toma las decisiones finales sobre el diagnóstico. Como resultado, el diagnóstico asistido, en el cual un sistema de proceso digital ofrece mayor evidencia e información al especialista, permite mejorar la calidad de su veredicto clínico [1].

Las ventajas relacionadas con los altos valores de velocidad, precisión y memoria para el almacenamiento de datos, que ofrecen los sistemas modernos de procesamiento digital permiten el desarrollo de sistemas de proceso de información sobre una vasta cantidad de datos médicos necesarios para el diagnóstico. De este modo, se incrementa la productividad y la eficacia del diagnóstico, tanto terapéutico, como de acciones preventivas por el uso económicamente justificado de computadores para resolver problemas médicos. No obstante, se debe hacer hincapié en que el diagnóstico asistido es siempre de carácter consultivo. Un computador no puede reemplazar el veredicto médico; sin embargo, puede sugerir una serie de decisiones con cierto nivel de

confiabilidad, de manera que sean finalmente aceptadas o rechazadas por el médico. El especialista utiliza la respuesta del computador como una segunda opinión, pero es el médico quien toma la decisión final.

En electroencefalografía clínica se utilizan diversas herramientas para mejorar el diagnóstico de algunas patologías mencionadas anteriormente que son incorporados a los programas de registro, y los cada vez más frecuentes programas de análisis de señales EEG. Las herramientas que proporcionan estos programas van desde simples cambios de escala para visualizar fenómenos de corta o larga duración, pasando por la aplicación individual o general de filtros digitales, detección y extracción de artefactos (en conjunto, todos estos se conocen como preprocesamiento de la señal), hasta algoritmos de extracción de características y clasificación.

Por último, cabe anotar que en concordancia con el número de trabajos presentados, relacionados con la investigación y desarrollo de aplicaciones de diagnóstico asistido en *IEEE EMB Society*, *Biosignals* y las publicaciones en *IEEE Transactions on Biomedical Engineering* del 2000 al 2007 (Electrocardiografía - 210, Fonocardiografía - 73), se observa que la mayoría de los trabajos se ha centrado en el área de preproceso y extracción/selección de características orientadas al reconocimiento de estados funcionales.

1.8.1 Sistema de Reconocimiento de Patrones

Usualmente un sistema de reconocimiento de patrones se describe en concordancia con el diagrama que se aprecia en la Figura 5. En una primera etapa, a partir del objeto (observación) se extraen mediciones (señales capturadas por medio de sensores) que se deben revisar y adecuar a fin de reducir o descartar problemas derivados del ruido o falla en los instrumentos de medida. A partir de las señales capturadas y adecuadas, se inicia la generación de características, que permite construir valores representativos que revelen o permitan descubrir algún tipo de patrón en los objetos que se analizan. Una vez construido el conjunto de características es necesario realizar el preproceso, con el fin de disminuir la influencia de los errores de registro sobre los datos caracterizados. Posteriormente, es posible hacer adaptaciones y transformaciones de dicho conjunto, de tal manera que se resalte el patrón subyacente en los objetos por medio de técnicas de selección o extracción de características. Finalmente, en la etapa de clasificación, es donde se hace una asociación del objeto a un tipo de clase (Daza-Santacoloma, 2007). La clasificación requiere ser validada y afinada, a fin de obtener un sistema con capacidad de generalización y precisión de respuesta [24].

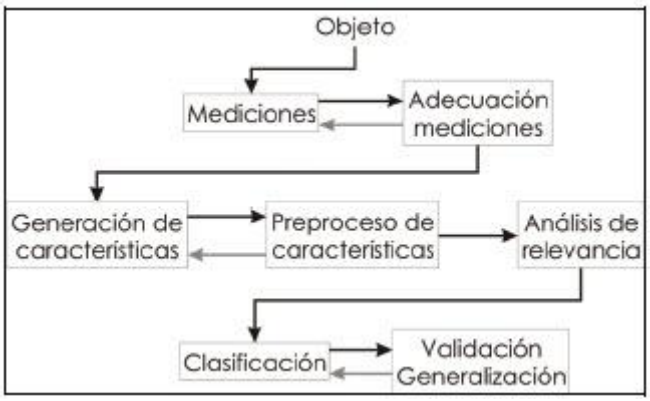


Figura 5. Sistema de reconocimiento de patrones

1.9 ASPECTOS TÉCNICOS DEL REGISTRO Y ANALISIS DE SEÑALES EEG

En la Figura 6 puede observarse un sistema moderno de registro y análisis EEG utilizado por los especialistas para el diagnóstico médico. A continuación se expondrán algunas de las características más importantes a tener en cuenta en cada uno de estos componentes.

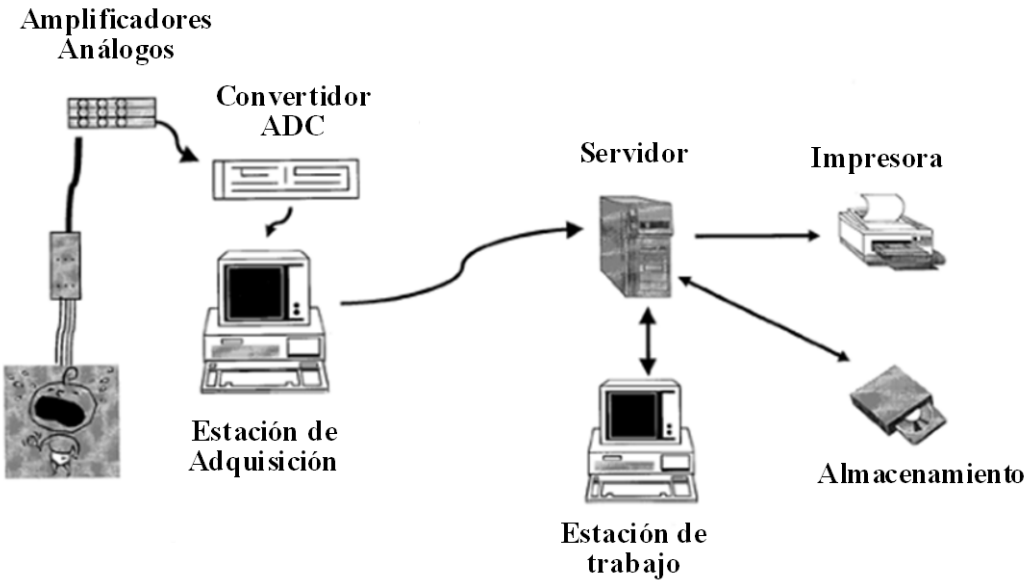


Figura 6. Sistema de registro y análisis de señales EEG típico.

1.10 TIPOS DE ELECTRODOS

Los electrodos para el registro apropiado de las señales EEG son cruciales para tener una buena calidad en los datos adquiridos. Existen diferentes tipos de electrodos dependiendo de las necesidades y el tipo de registro:

- Desechables (sin gel, con gel incluido).
- Electrodos de disco reusables (oro, plata, etc).
- Gorro de electrodos.
- Electrodos con base salina.
- Electrodos de aguja (para registros intracraneales).

La característica común más importante que deben poseer los electrodos es una impedancia menor a 5kΩ y tener un balance en ±1kΩ entre los electrodos. Algunos equipos tienen ganancia programable y un sistema de calibración que permite ajustar dichas ganancias de acuerdo a la señal de respuesta del canal a una señal de voltaje conocida [25].

1.10.1 Montaje de Electrodos

Un comité de la Federación internacional de Sociedades de Electroencefalografía y Neurofisiología Clínica recomendó un sistema específico de posicionamiento y denominación de electrodos para usar en todos los laboratorios bajo condiciones estándares, como se observa en la Figura 7 (Jasper, 1958). Esta recomendación se conoce hoy en día como el estándar 10-20. Cada canal recibe un nombre (una o dos letras y un número) de acuerdo a la zona de la cabeza donde se encuentra ubicado. Para determinar la posición en un lugar específico de la cabeza de cada paciente se deben hacer algunas mediciones longitudinales y transversales de las dimensiones craneales.

En 1991 la Sociedad Americana de EEG aumentó a la guía de nomenclatura de ubicación de electrodos, hasta llegar a la identificación de 75 posiciones de electrodos, tal como aparece en la Figura 8.

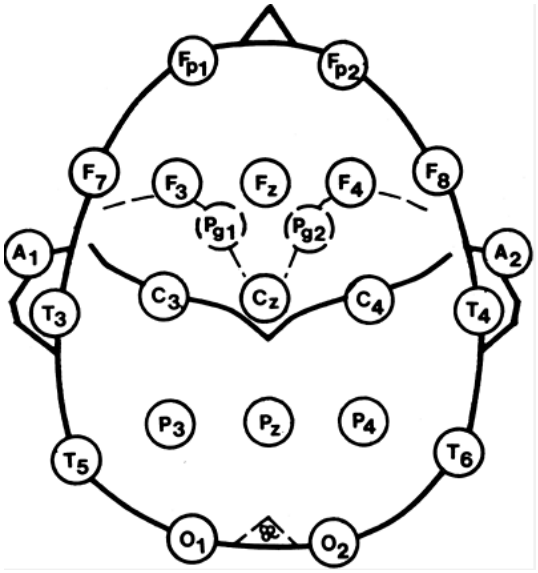


Figura 7. Sistema Internacional 10-20 de ubicación de electrodos

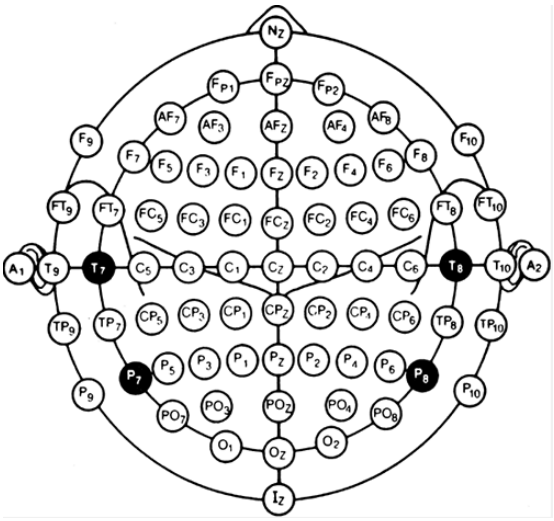


Figura 8. Sistema aumentado de posicionamiento de electrodos.

Los montajes de EEG normalmente se clasifican en dos categorías, por un lado están los montajes que utilizan una señal común (denominados monopolares) a la cual están referidos cada uno de los canales y por el otro están los montajes que hacen referencia a registrar la diferencia de potencial entre dos canales, a este último tipo de montaje se les denomina montajes bipolares (Ver Figura 9). Si bien el hardware habitualmente posibilita el registro directo de las señales bipolares, una forma más práctica de obtener estos registros es mediante el software de visualización, el cual debe permitir visualizar el registro resultado de restar parejas de señales monopolares para obtener las señales bipolares respectivas, a este tipo de arreglos se les denomina montajes por software.

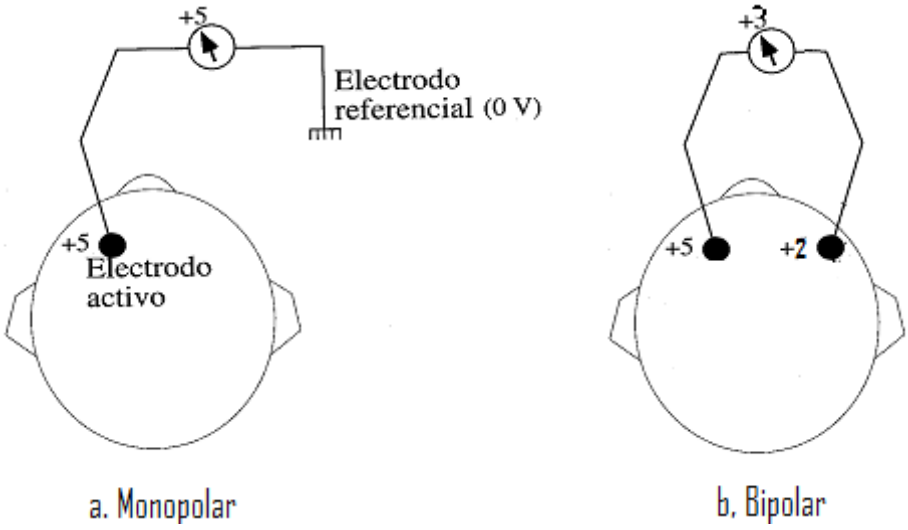


Figura 9. Sistemas de registro (a) Monopolar y (b) Bipolar

1.11 SEÑALES ADICIONALES AL REGISTRO DE EEG

En algunos registros de EEG se puede adicionar un canal de ECG para correlacionar la actividad eléctrica del corazón con la actividad eléctrica del cerebro, e incluso en algunas ocasiones también se correlaciona con la actividad electromiográfica (EMG) en algún tejido muscular en particular. Cada vez es más frecuente acompañar al registro de EEG con el video que registra la actividad del paciente para determinar señales de ruido debidas a movimientos del paciente, pero también es útil para visualizar la actividad del paciente en eventos especiales (por ejemplo, en episodios epilépticos). Es muy importante que la visualización de las señales este sincronizadas en el tiempo con la señal del video.

1.12 ESPECIFICACIONES DEL REGISTRO DE SEÑALES

1.12.1 Escalas de tiempo y voltaje

La duración y la cantidad de señales de un registro EEG depende del tipo de examen que el médico especialista desee realizar de acuerdo con la sintomatología que presente el paciente, algunos registros pueden durar diez minutos y en algunos otros varias horas, y la cantidad de señales involucradas depende del montaje elegido, y puede ser de 28 hasta 128 electrodos. Por estas razones el software de visualización debe tener la posibilidad de variar las escalas de voltaje y tiempo para que el especialista pueda observar con diferente nivel de detalle las diferentes partes del registro.

1.12.1.1 Muestras x Segundo

Las frecuencias que hacen parte de la información relevante contenida en un registro de EEG indican que el muestreo debe hacerse a mínimo 128 Hz, pero para ciertas patologías este valor puede llegar a ser de 512 Hz. Inclusive, en algunos registros intracraneales, donde se pretende conocer una información más localizada, se trabaja con señales del orden de 40 kHz, pero para este tipo de registro normalmente se utiliza otro tipo de herramientas de visualización y análisis [1].

En el análisis visual de las señales de EEG es común emplear diferentes escalas de tiempo (en los registros no automatizados se hablaba de velocidad rápida y lenta del papel), en escalas donde el tiempo está comprimido se pueden ver fenómenos repetitivos con períodos de repetición elevados; mientras que el análisis de fenómenos de corta duración, son más fácilmente detectables en una escala donde un lapso corto de tiempo ocupe un área considerable. Por estas razones el software de visualización y análisis deberá permitir variar la escala de tiempo por parte del usuario.

1.12.1.2 Cantidad de bits/sensitividad

Las señales de EEG tienen amplitudes del orden de los microvoltios. Para capturar la información efectiva de estas señales se debe amplificar antes de pasar por el ADC. Entre más bits tenga el ADC mayor será la resolución en voltaje, típicamente en los registros de EEG se utiliza un ADC de 12 bits o 16 bits, mayor cantidad de bits implican mayor consumo de memoria RAM en el momento de captura y de espacio en el disco para su almacenamiento. Una vez se tienen las muestras en la memoria y/o en el sistema de almacenamiento permanente es conveniente poder cambiar la escala de voltaje de cada señal por separado para poder resaltar las características individuales de cada una de ellas en el momento que el especialista lo considere conveniente, e inclusive en algunos momentos es útil no presentar en pantalla algunas señales para poder visualizar apropiadamente las otras.

1.12.2 Filtros

Se pueden aplicar filtros antes o después del ADC (inclusive se emplean filtros de software que se configuran en el programa de visualización). Los filtros deben ser diseñados de tal forma que no generen distorsión alguna sobre la señal. Los filtros pasa altos tienen una frecuencia de corte mínima de 0.5 Hz y se utilizan para remover las perturbaciones de baja frecuencia como la respiración.

Por otro lado las componentes de alta frecuencia son eliminadas con filtros pasa bajos de aproximadamente 50-70 Hz.[25] . También, son muy empleados los filtros *notch* a la frecuencia de la red eléctrica (60 Hz).

1.12.3 Eventos

Durante la toma de las señales del registro EEG ocurren ciertos eventos que es importante tener en cuenta en el momento del análisis. Por ejemplo cuando el paciente mueve los ojos aparece una señal DC de gran amplitud en las señales de la región frontal, y el especialista o el software de análisis deberá distinguir este evento con otro tipo de fuente que pudiera ocasionar dicho nivel DC. También es habitual que durante el registro en pacientes con epilepsia se le induzcan episodios epilépticos mediante la administración de cierta droga o la exposición a luces estroboscópicas y es importante tener en cuenta el momento en que se realizaron tales acciones.

Por esta razón además de las señales biológicas en el registro EEG se agregan una serie de marcas que indican la ocurrencia de estos y otros eventos especiales. Otras marcas pueden ser agregadas por los especialistas o por el software de análisis para indicar la ocurrencia de ciertos patrones asociados con actividad anormal en cierta posición en el tiempo del registro para que luego el especialista a partir del análisis de la frecuencia de ocurrencia y las zonas cerebrales implicadas pueda hacer un diagnóstico más acertado.

1.12.4 Almacenamiento de los registros. Formatos de archivos EEG digitales, Compatibilidad entre sistemas

Cuando las empresas que desarrollan equipos de EEG fueron migrando hacia las versiones controladas con PC, no acordaron un formato único para los archivos que contienen las señales de EEG. Razón por la cual cada empresa se "inventó" su propio formato: cada desarrollador de software de captura y/o análisis de EEG desarrolló sus protocolos de forma aislada (tal vez por razones comerciales), y no era posible utilizar una herramienta de visualización o diagnostico de una empresa con registros tomados en equipos otro proveedor, llegando a darse el caso que archivos de diferentes equipos de una misma empresa no eran compatibles entre sí [26].

Se han hecho algunos esfuerzos por crear un formato estándar, pero estos no han tenido la acogida necesaria por parte de las empresas, como ejemplo el formato de archivo propuesto por la ATSM (*American Society for Testing and Material*) en 1993. Era un formato ASCII que además de los registros podía contener las anotaciones médicas, fue aceptado por muy pocas empresas (posiblemente por el hecho de ser ASCII, genera archivos demasiado voluminosos y de más lento el acceso comparados con los formatos binarios) [27].

2. ANTECEDENTES

En este capítulo se presenta una breve historia de la evolución del procesamiento digital de señales (DSP) y las diferentes herramientas computacionales utilizadas para la implementación de algoritmos de procesamiento de señales en general y en particular de señales EEG. Luego, una sipnosis del Estado del Arte en el mundo. Posteriormente, se presentarán algunos desarrollos en Colombia y más específicamente en la región del eje cafetero. Al finalizar se dará una descripción de la motivación que se tuvo para la formulación de éste proyecto.

El tratamiento de señales en tiempo discreto ha avanzado con pasos desiguales durante un largo periodo de tiempo. Hasta principios de los años cincuenta el tratamiento de señales se realizaba con circuitos electrónicos o incluso con dispositivos mecánicos. Aunque los computadores digitales ya estaban disponibles en entornos de negocios y en laboratorios científicos, éstos eran caros y de capacidad relativamente limitada.

Aunque el procesamiento de señales mediante computadores digitales ofrecía tremendas ventajas de flexibilidad, el procesado no se podía realizar en tiempo real. Uno de los primeros usos fue en la prospección petrolífera, donde se grababan los datos sísmicos en cintas magnéticas para su procesamiento posterior.

Las aportaciones de Cooley y Tukey (1965) de un algoritmo eficiente para el cálculo de las transformadas de Fourier aceleró el uso del computador digital. Muchas aplicaciones desarrolladas requerían del análisis espectral de la señal y con las nuevas transformadas rápidas se redujo en varios órdenes de magnitud el tiempo de cómputo. Además, se dieron cuenta de que el nuevo algoritmo se podría implementar en hardware digital específico, por lo que muchos algoritmos de tratamiento digital de señales que previamente eran impracticables comenzaron a verse como posibles.

Otro desarrollo importante en la historia del Procesamiento de Señales ocurrió en el terreno de la Microelectrónica. Aunque los primeros microprocesadores eran demasiado lentos para implementar en tiempo real la mayoría de los sistemas en tiempo discreto, a mediados de los ochenta la tecnología de los circuitos integrados había avanzado hasta el nivel de permitir la realización de microcomputadores en coma fijo y coma flotante con arquitecturas especialmente diseñadas para realizar algoritmos de procesamiento de señales en tiempo discreto. A estos procesadores se les conoce por el acrónimo de DSP (*Digital Signal Processor*). Con esta tecnología llegó, por primera vez, la posibilidad de una amplia aplicación de las técnicas de tratamiento de señales en tiempo discreto. Apoyados en estos desarrollos se pasó a los diseños de los microprocesadores con parte de su arquitectura especializada en tareas de procesamiento de señales, sea el caso más llamativo el conjunto de instrucciones MMX insertadas en la familia INTEL a partir del mítico PENTIUM[28].

A nivel de software la evolución en la implementación de algoritmos, además de los avances en los propios algoritmos, va de la mano con la evolución de los lenguajes. En un principio se implementaron algoritmos en lenguaje FORTRAN. Y si bien estrictamente hablando, FORTRAN no fue el primer lenguaje de programación que existió (Laning & Zierler, del MIT, ya tenían un

compilador de un lenguaje algebraico en 1952), sí fue el primero en atraer la atención de una parte importante de la reducida comunidad de usuarios de computadoras de la época. El desarrollo de FORTRAN comenzó en 1955 y el lenguaje se liberó finalmente en abril de 1957.

Luego surgieron otros lenguajes como ALGOL, LISP (1958), COBOL (1960), Basic (1964), pero FORTRAN siempre tuvo la preferencia en cuanto a la implementación de algoritmos sobre señales, pues su compilador fue creado para optimizar principalmente las operaciones de tipo matemático. Luego el Lenguaje C en 1973 es creado por Dennis Ritchie en los laboratorios Bell. Tuvo mucho éxito al ser utilizado para el desarrollo del sistema operativo Unix. Ya en 1983 : Bjarn Stroustrup desarrolla una extensión orientada objeto al lenguaje C : el C ++. Sin embargo el gran empuje en el procesamiento digital de señales ha sido en parte debido al surgimiento de herramientas que permiten hacer mucho más fácil la tarea de codificar los algoritmos, en particular la herramienta Matlab®, de la que se hablará próximamente.

A nivel de algoritmos uno de los primeros avances formales en DSP fue el artículo "*Certain topics in TelegraphTransmission Theory*", publicado por Harry Nyquist en 1928, en el cual se presentó el efecto producido en el espectro de frecuencia de una señal análoga al ser discretizada en el tiempo, y se planteó que, para preservar la información original, la tasa de muestreo debía ser mayor que el doble de la máxima componente de frecuencia contenida en la señal análoga. Posteriormente, en 1949, Claude Shannon publicó el artículo "*Communications in the Presence of Noise*", donde demostró que es posible reconstruir perfectamente una señal análoga a partir de sus muestras, si se dispone de un filtro pasabajos análogo ideal. (Si bien no es posible fabricar un filtro de este tipo, es posible aproximarse bastante a él en muchas situaciones prácticas) [29].

Además se debe tener en cuenta las ya mencionadas aportaciones de Cooley y Tukey (1965) en la implementación de la FFT. Aunque las referencias históricas enfatizan en los inicios con el desarrollo de algoritmos de procesamiento en línea como la aplicación de los filtros digitales: Al principio, fueron llamados "filtros numéricos" o "filtros de datos muestreados". Más tarde se les llamó "Filtros digitales". Su nombre fue unificado después 1965 (J. Kaiser F. 1965). También surgieron aportes de la teoría de comunicaciones: Realizaciones de sistemas PCM: T1-system (USA) --- 1960. PCM-24 system (Japan) --- 1965. Igualmente se deben mencionar los algoritmos de análisis numérico: solución de sistemas de ecuaciones, ecuaciones diferenciales, ecuaciones integrales, etc.

Pronto el procesamiento digital de señales se convirtió en un área multidisciplinaria en la cual están inmersas varias disciplinas del saber: como Ingeniería, Ciencias Naturales, Medicina, agricultura, etc. Las áreas de la ingeniería involucradas en el procesamiento de señales son: Ingeniería eléctrica y electrónica, tecnologías de la computación y la información, Tecnología de la Comunicación, Ingeniería de control, Ingeniería de sistemas. [30].

Matlab® fue creado en 1984 por la empresa MathWork, surgiendo la primera versión con la idea de emplear paquetes de subrutinas escritas en Fortran en los cursos de álgebra lineal y análisis numérico, sin necesidad de escribir programas en dicho lenguaje. El lenguaje de programación M fue creado en 1970 para proporcionar un sencillo acceso al software de matrices LINPACK y EISPACK sin tener que usar Fortran. En 2004, se estimaba que MATLAB® era empleado por más de un millón de personas en ámbitos académicos y empresariales. Matlab® ha evolucionado y

crecido con las aportaciones de muchos usuarios. En entornos universitarios se ha convertido junto con Matemática y Maple, en una herramienta instructora básica para cursos de matemática aplicada, así como para cursos avanzados en otras áreas.

En la adquisición y procesamiento de señales digitales es necesario destacar el surgimiento de la primera versión para Macintosh del programa LabView® de la empresa National Instruments en 1986, y sus posteriores versiones, incluyendo la primera versión para PC en 1992. La idea del instrumento virtual, en su momento revolucionaria, en la industria de medición y automatización y la tecnología, ayudó a los ingenieros y científicos a personalizar los sistemas de medición para adaptarse a sus necesidades. El lenguaje intuitivo de programación gráfica que se utiliza en LabView® ha sido la clave de la popularidad con los principiantes y los programadores con experiencia en aplicaciones de ingeniería para diversas industrias [31].

A nivel de señales EEG, en el procesamiento de este tipo de señales habría que remontarse a AL Loomis, Harvey ES, y Hobart GA quienes fueron pioneros que estudiaron matemáticamente los patrones y las etapas de sueño EEG en el ser humano en 1.937. Berger asistido por Dietch (1932) aplicó el análisis de Fourier a las secuencias de EEG. En un estudio titulado "*Una técnica de adición en caso de detección de señales pequeñas en un fondo irregular grande*", George D. Dawson del Hospital Nacional de *Queen Square*, en Londres demostró potenciales evocados a la estimulación eléctrica del nervio cubital (Dawson, 1951). Luego se implementaron las versiones computarizadas de dichos algoritmos en la década de 1.960. A finales de ésta década se veía ya que el análisis computarizado de las componentes en frecuencia había llegado para quedarse y llegaría a ser de enorme valor no sólo en la investigación psicofisiológica, sino también en la evaluación de los efectos neurofarmacológicos, (este avance fue gracias a la publicación del algoritmo de la FFT).

En los últimos 30 años los registros EEG han sido una invaluable fuente de información sobre todo para diagnostico y tratamiento de epilepsia, pero también se ha descubierto que el EEG está lleno de información para la gran mayoría de las enfermedades neurológicas. A tal punto fue despreciada esta información, que se llego a publicar: "Es muy lamentable que esta rica fuente de información haya sido infrautilizada en la mayoría de las instituciones de enseñanza y hospitales neurológicos en general" [32]. Sin embargo, hay que anotar que a nivel teórico hubo un gran avance en la comprensión de este tipo de señales con el advenimiento de técnicas multi-resolución, que a pesar de haber tenido sus inicios en la década de 1.920, tuvo su apogeo a finales de la década de 1.980 e inicios de 1.990 con las publicaciones de Mallat, Daubechies y otros. Debido a estos aportes, a la internet y otros avances, el análisis de señales biológicas y en particular de señales EEG tuvo un empuje extraordinario, por mencionar sólo algunos de sus efectos, ya en el año de 1.999 aparece el primer *toolbox* de Matlab® dedicado señales biológicas (NIH CORTEX Toolbox), luego vienen muchos más dedicados a señales biomédicas (BrainsStrom- 2000, PRANA-2002, EEGLAB-2004, BrainMaps 2006, etc.).

En el año 2004 aparece el artículo libRASCH: *A Programming Framework*, publicado por R. Schneider en la revista *Computers in Cardiology* de la IEEE [33]. En dicho artículo el autor propone un sistema de librerías que permiten el acceso a los diferentes formatos de datos para los registros de señales biológicas, y un mecanismo para que puedan incorporarse a diferentes lenguajes

de programación por medio de *plugins*, y con el mismo mecanismo, una forma de incorporar también rutinas de procesamiento sobre dichas señales.

También en el año 2004 el centro de Cómputo para Neurociencias de la Universidad de California desarrolló el EEGLAB, un ToolBox de Matlab® que, según su página oficial, tiene más de 40.000 descargas en los últimos 5 años. La descripción de este *toolbox* es: un conjunto de herramientas y la interfaz gráfica de usuario, EEGLAB, que corre bajo multiplataforma en el entorno MATLAB® (*The Mathworks, Inc*.) para el procesamiento de colecciones o de una señal y / o un promedio de datos de EEG de cualquier número de canales. Las funciones disponibles incluyen importación de datos EEG, voltaje de cada canal e información sobre eventos, visualización de datos (desplazamiento, mapas topográficos de la corteza cerebral y trazado de modelo de dipolo), pre-procesamiento (incluido el rechazo de artificios, filtrado, selección de época, y un promedio), análisis de componentes independientes (ICA) y tiempo / frecuencia, incluida la descomposición de componentes de coherencia cruzada de cada canal con el apoyo de métodos estadísticos *bootstrap* basados en re muestreo de los datos.

Las funciones de EEGLAB están organizadas en tres capas. Las funciones de la primera capa sirven para interactuar con los datos a través de la interfaz gráfica sin necesidad de usar la sintaxis de MATLAB®; las opciones de menú asistidos, para afinar el comportamiento de EEGLAB a la memoria disponible; las funciones de la capa del medio permiten a los usuarios personalizar el procesamiento de datos utilizando el historial de comandos y funciones interactivas 'pop'. Los usuarios con experiencia en MATLAB® pueden utilizar las estructuras de datos EEGLAB para escribir rutinas de procesamiento de señales propias y/o secuencias de comandos de análisis a partir de scripts de Matlab®[34].

En la actualidad mundial, y a nivel de investigaciones académicas cada día se desarrollan técnicas que ayudan al diagnóstico, como son: los métodos de eliminación de artificios con aplicación de regresión en el dominio del tiempo [6], o en el dominio de la frecuencia; el uso de filtros espaciales [7] y análisis de coherencia de fase y sincronía. También se destacan algunos trabajos que permiten aplicar métodos matemáticos en el estudio de fuentes epilopatogénicas para la detección y predicción de crisis. En estos, primero se establece una solución al problema directo y, a partir de éste, se halla la solución al problema inverso [8]. También se utiliza con más frecuencia la superposición de actividad eléctrica y otras técnicas de neuroimagen, para tener una mejor resolución espacial [9].

En lo referente a BCI, el número de investigadores ha crecido rápidamente en la última década, y los avances se han dado de manera acelerada gracias a una acción interdisciplinaria de científicos, ingenieros y médicos de diferentes disciplinas que han enfrentado un buen número de problemas cruciales. Neurosky [10] y Emotiv [11] son dos ejemplos claros de cómo algunas empresas han enfocado sus esfuerzos para aprovechar los potenciales eléctricos de la corteza cerebral, en el control de dispositivos electrónicos. Ambas empresas ofrecen diademas con electrodos secos con los cuales se detectan las señales cerebrales y ofrecen kits de desarrollo de software (SDK, por sus siglas en inglés *Software Development Kit*), para investigadores y desarrolladores de software.

Por ahora, los esfuerzos de estas empresas están concentrados en la simplificación de las interfaces en los juegos de video, pero son innegables las aplicaciones prácticas, de todo tipo, que se desprenden de estos primeros resultados: desde la asistencia a personas con limitaciones físicas,

brindándoles autosuficiencia en sus desplazamientos; pasando por nuevas formas de comunicación (acaso telepatía asistida por dispositivos electrónicos?); hasta nuevas formas de enfrentar al contrincante en las guerras futuras, donde no se precisará de la presencia física de un tripulante en máquinas de combate: La telequinesis está *ad portas* de nuestra vida

En Colombia se han hecho algunos avances en procesamiento de señales bioeléctricas, como lo demuestran la cantidad de congresos y publicaciones Nacionales y Latinoamericanas. En la implementación de los algoritmos de procesamiento de señales, sin embargo, se adolece de la falta de codificación en software de aplicación, pues en la mayoría de las ocasiones sólo se ha limitado a la utilización de *ToolBox* en Matlab® para probar el funcionamiento del algoritmo y cuando se llevan a aplicaciones a menudo usan sistemas de instrumentación virtual como LabView® que requieren para su uso legal de licencias de software cuyos costos son elevados. A continuación presentamos los trabajos más destacados en el desarrollo de entornos con bioseñales y algunos de EEG en particular.

En el año 2007, en la Universidad de los Andes se desarrolló e implementó un sistema basado en instrumentación virtual para la adquisición y procesamiento de señales fisiológicas para ser empleado como herramienta para mejorar el proceso de enseñanza-aprendizaje que se lleva a cabo en los laboratorios de fisiología de las facultades de Medicina de nuestro país. El hardware del sistema comprende un amplificador de bio-potenciales de cuatro canales, una tarjeta de adquisición de datos y un computador personal (PC) que permite la interface con el usuario. Para el desarrollo del software se empleó el lenguaje de programación gráfico LABVIEW® 6.1. [35]

En el año 2007, en la Universidad del Quindío se desarrolló un hardware que permite la adquisición, procesamiento y visualización de las 12 derivaciones estándar de un electrocardiograma (ECG) usando LabView®. El sistema consta de una etapa de instrumentación conectada a un cable para electrocardiograma que toma los potenciales de diez electrodos superficiales; con ellos, se adquieren tres derivaciones bipolares y seis derivaciones precordiales. Las nueve derivaciones fueron filtradas usando dos filtros análogos Butterworth de cuarto orden, pasa altos y pasa bajos, en cascada y luego fueron adquiridas a través de una tarjeta de adquisición de datos USB NI-6215. Finalmente, las derivaciones fueron filtradas digitalmente, visualizadas y almacenadas utilizando LabView®. [36].

En el año 2008, la línea de Ingeniería Biomédica dentro del programa de Ingeniería Electrónica de la UPB-Bucaramanga utilizando las facilidades ofrecidas por las herramientas hardware (DAQ, chasis) y software (*LabView*®) de National Instruments, se integró bajo una misma plataforma, y con una orientación pedagógica, el estudio del procesamiento de señales del cuerpo humano. Es así como surgió BIOLAB concebido como un laboratorio conformado por cuatro módulos didácticos para la adquisición y acondicionamiento de señales cardíacas (BIOLAB-ECG), señales cerebrales (BIOLAB-EEG y BIOLAB-STIM) y señales musculares (BIOLAB-EMG) [37].

En la zona cafetera se han hecho algunos trabajos dispersos en el área de las bioseñales, y en los últimos años se han sumado esfuerzos para desarrollar trabajos de investigación conjuntos entre las universidades públicas de la región e inclusive con la empresa privada, alianzas que han permitido capacitar material humano por medio de programas de maestrías y doctorados, prueba de esto son los proyectos de investigación: “Detección de Fuentes Epileptopatogénicas en señales de EEG”.

Sistema Automatizado de clasificación de eventos fisiológicos a partir de patrones fisiológicos como soporte en el tratamiento de la enfermedad de Párkinson y otros desordenes Neurofisiológicos UTP. Dentro de los cuales se han logrado desarrollar varios proyectos de grado a nivel de Pregrado, de Maestría y de Doctorado, por ejemplo: Análisis de la Variabilidad del Intervalo QT en la señal Electrocardiográfica. Edison Duque. Localización de fuentes electroencefalográficas empleando modelos inversos distribuidos basados en norma mínima. Victoria Eugenia Montes Restrepo. Universidad Tecnológica de Pereira 2009.

Neurocentro, el Instituto de Epilepsia y Parkinson del Eje Cafetero, ha realizado en los últimos años alianzas estratégicas con universidades de la región y producto de éstas alianzas se han desarrollado investigaciones conjuntas en el campo de la neurología, aportando, además de especialistas en el campo, los registros que sirven como base de datos para el entrenamiento de algoritmos.

El Instituto Neurocentro utiliza el equipo **Bio-logic Ceegraph EEG** [38] de la empresa Biologic-System el cual dispone de módulos de adquisición hasta de 132 canales con la respectiva circuitería de adquisición, filtrado y amplificación de *Headboxes* de diferentes configuraciones (Gamma, Netlink, XL o XL2), que permite adquirir las señales de EEG, opcionalmente se pueden agregar al registro un canal de ECG, Video del paciente y marcas de estimulación óptica (generada automáticamente con el equipo) y algunas otras marcas producidas por el equipo y/o la persona que realiza el registro. Además del registro el programa permite visualizar en diferentes escalas de tiempo y voltaje las señales y los eventos producidos durante un registro. Pero el procesamiento de los datos se limita a filtros digitales configurables por el usuario que pueden activarse o desactivarse durante o después del registro.

Para el análisis de las señales que sirven para el diagnóstico los Neurólogos del Instituto utilizan, entre otros, el paquete de software ***Persyst Insight*** [39] que además de las opciones de visualización y filtros que posee el Bio-logic Ceegraph EEG, permite algunas mediciones básicas de amplitud (máxima/mínima), frecuencia y/o correlación de señales en el rango que escoja el usuario, de igual manera, dispone de una serie de *Plugins* que permiten realizar diferentes análisis (detección de *Spikes*, *Spikes-Brust*, marcas configurables), y visualizaciones (Video, mapas de niveles de voltaje, espectrogramas, etc).

Si bien hoy en día se encuentran diversos tipos de aplicaciones que contemplan lo antes dicho, algunas aunque muy costosas no permiten realizar ningún tipo de procesamiento o análisis digital a las señales EEG, y si lo permiten, las librerías o herramientas de análisis tienen un costo bastante significativo o requieren de un especialista en software para poderlas incluir, esto motivo a los autores a desarrollar una herramienta que permitiera visualizar los registros EEG de la manera estándar, es decir, que se pudiera fácilmente variar las escalas de tiempo, los niveles de voltaje bien sea de uno o de varios registros simultáneamente, observar el video, sincronizado con la señal de los registros, y sobre todo que quedará abierta la posibilidad para incluir fácilmente las librerías o *plugings* desarrollados en Matlab® o en C++ para el procesamiento de las señales.

Actualmente se tiene implementado mediante este tipo de mecanismo de librerías (Arquitectura abierta) los algoritmos correspondientes a los filtros digitales y, en lo referente al preprocesamiento, procesamiento y análisis de EEG ya se tiene implementados en Matlab® algunas rutinas que se

encuentran en proceso de conversión o adaptación al sistema las cuales, en su momento podrían ser modificadas, o incluidas, fácilmente.

3. ARQUITECTURA DEL VISOR

En éste capítulo se presentará la forma como fue construida la aplicación, se enumerarán sus componentes principales y se hará una breve descripción de su funcionamiento. El nombre dado a la aplicación fue VISOR EEG_M, con código de registro ante la dirección nacional del derecho de autor, numero 12-26-263 (Libro-Tomo-Partida).

El Visor EEG_M es una aplicación que consta de una serie de componentes gráficos y lógicos que se encargan de acceder y visualizar las señales tomadas de registros de EEG en diferentes formatos, y una interface para comunicarse con módulos externos con los cuales se les puede aplicar diferentes algoritmos para el procesamiento de señales utilizando diversos lenguajes de programación.

El visor EEG_M está compuesto de un programa principal y una serie de módulos independientes que se cargan dinámicamente durante la inicialización del programa y que se relacionen con éste por medio de funciones y controles que son declarados en el programa principal (o un módulo binario) y definidos en los módulos, como se ilustra en la Figura 10. El código escrito en los módulos adicionales tiene tres propósitos:

- Agregarle nuevas capacidades al programa para que el usuario final las pueda utilizar por medio de los controles, o

- Agregar nuevas funciones para que en una parte del programa se puedan invocar nuevas funciones para un procedimiento ya establecido, o

- Crear un procedimiento para que invoque a funciones escritas en otros módulos o en el programa principal.

Un módulo consiste en una sección de programa creada y/o compilada de forma independiente del VisorEEG_M, pero que puede incorporarse a éste, si cumple con los prototipos definidos en las APIs (*Aplication Program Interfaces*), creadas para tal propósito. Existen dos tipos de módulos: los módulos binarios y los módulos interpretados, dependiendo de la forma como se relacionan con el programa. Los módulos binarios son aquellos que tienen código ejecutable directamente en la CPU, mientras que los módulos interpretados, como su nombre lo indica, requieren de un módulo intérprete para su ejecución. Los intérpretes (que a su vez deben cumplir con otras APIs del Visor), también se cargan de forma dinámica de tal manera que pueden agregarse nuevos intérpretes sin necesidad de recompilar la aplicación.

Para hacer posible esta comunicación del programa principal con los módulos se utiliza una serie de clases que permiten cargar, inicializar y ejecutar apropiadamente las funciones de los módulos, ya sea directamente o a través de los intérpretes. En una sección posterior se explicarán los componentes del programa principal que sirven para dicha comunicación y en el Anexo 1 se presenta una descripción detalla de la forma de construir estos módulos.

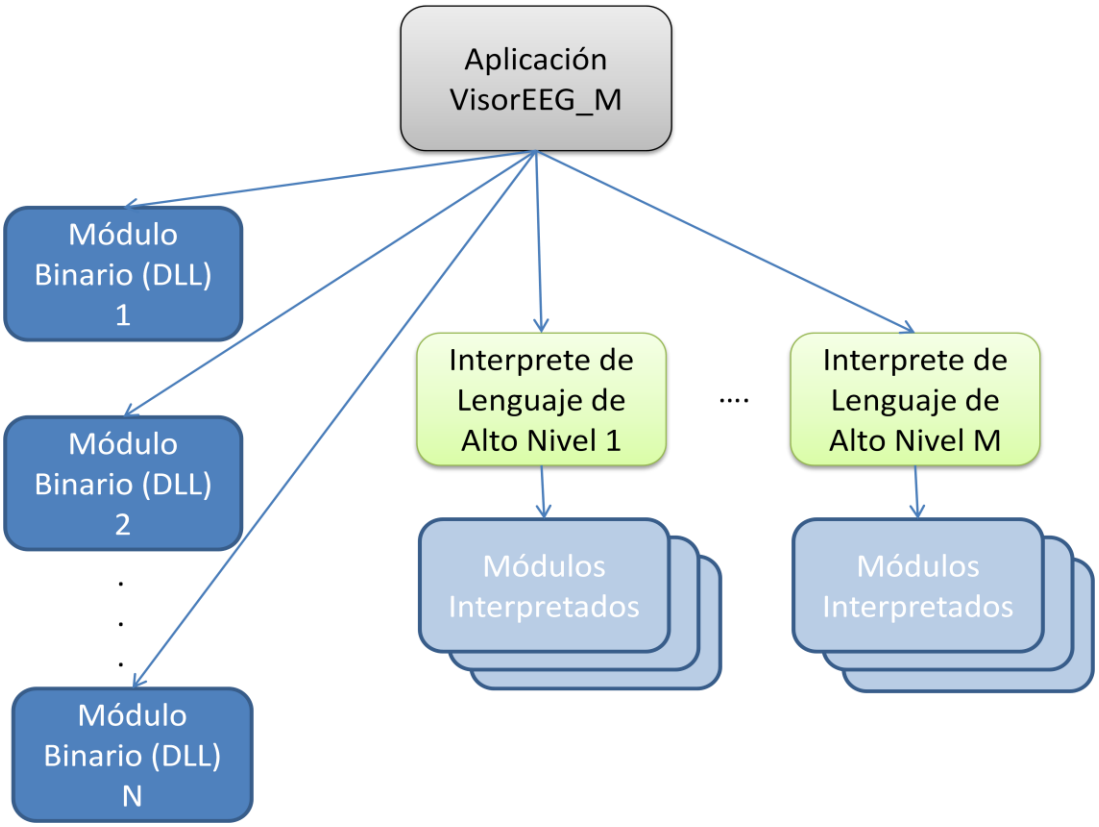


Figura 10. Diagrama de bloques del Sistema Modular del Visor EEG

En un archivo de configuración del programa se guardan los controles que han sido instalados en el programa por cada módulo; cada vez que se inicializa la aplicación, y una vez se cargan los módulos, se agregan dichos controles a la barra de herramientas del programa. En este archivo de configuración también se guardan los procedimientos con la respectiva función asociada y de forma similar son actualizadas cada que se inicializa la aplicación.

3.1 APLICACIÓN VISOR EEG_M

El funcionamiento del programa se puede dividir en dos partes: la primera va desde el inicio hasta cuando queda listo para cargar un archivo de señales, y la segunda, va desde este punto hasta el final. La primera parte aparece en la Figura 11.

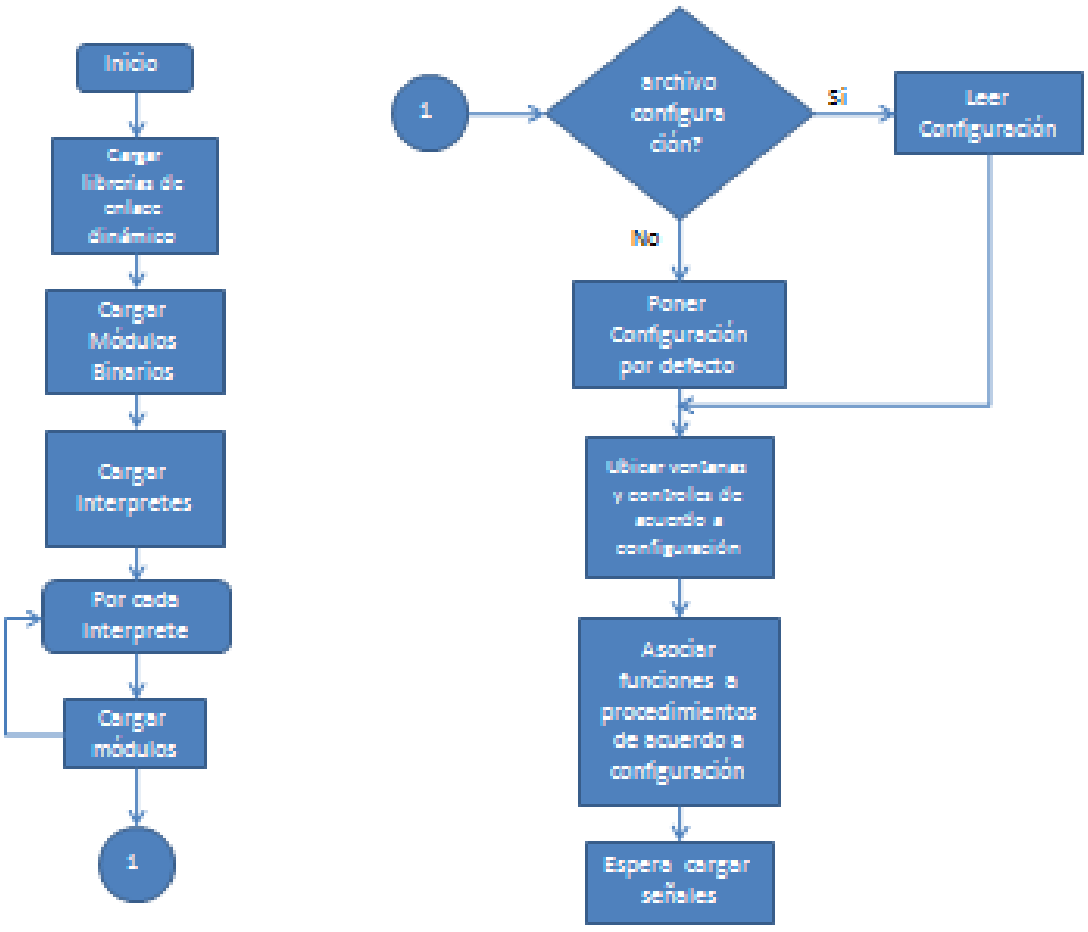


Figura 11. Diagrama de flujo del programa hasta que queda listo para cargar señales

Cuando el programa inicia, lo primero que hace es cargar las librerías DLL que le permiten acceder a los registros y al video asociado, que son respectivamente: ***librasch.dll*** y ***libvlc.dll***. Luego se cargan cada uno de los módulos binarios, que son archivos DLL con funciones, procedimientos y controles que se han adicionado al programa, y que se encuentren en la carpeta ***modulos****;* por cada módulo se hace la respectiva validación de que posea las funciones declaradas en las APIs del programa. Inmediatamente después, el programa busca cada una de las subcarpetas contenidas en la carpeta ***modulos***, para ver si contienen módulos interpretados, buscando que dichas carpetas contengan el archivo *soporte.dll*; si lo encuentra, lo carga (previa validación de las respectivas APIs). Posteriormente, utilizando cada uno de estos archivos de soporte, busca y carga todos los módulos interpretados de cada lenguaje, utilizando las funciones definidas en ***soporte.dll*** que se encargan de cargar los módulos en memoria. Después de cargar los módulos, el programa continúa con la lectura del archivo de configuración; si no existe, utiliza una configuración por defecto. Con esta configuración, el programa determina cuales ventanas son visibles, así como su posición y

tamaño. La última parte de la configuración la utiliza para asociar las funciones a los respectivos procedimientos, con la debida validación de parámetros y valores de retorno (proceso que será explicado con más detalle en este capítulo). Una vez terminado este proceso, el programa se encuentra listo para cargar un archivo de señales. El diagrama de flujo a partir de este punto se explica en detalle en la Figura 12.

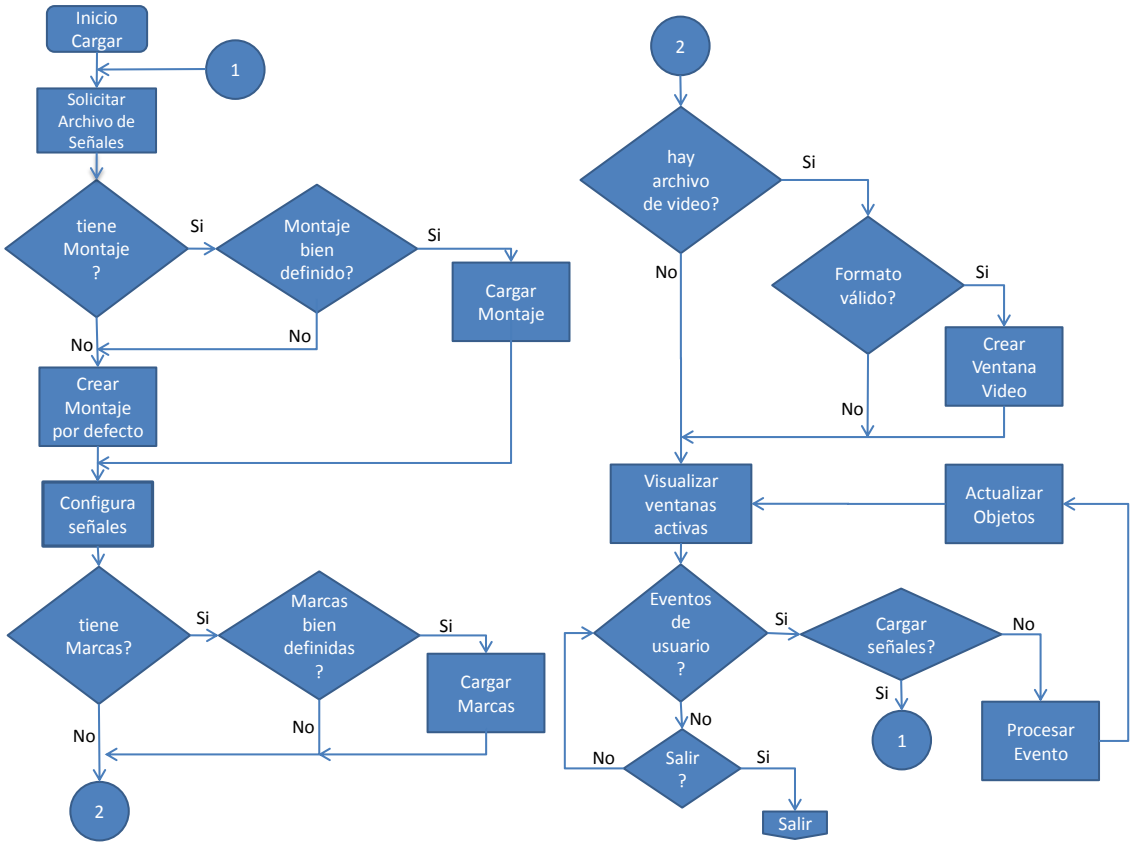


Figura 12. Diagrama de flujo del programa una vez se elige cargar un archivo de señales

Cada vez que se selecciona ***Abrir*** un archivo de señales, el programa despliega un caja de diálogos, donde se muestran todos los archivos con el formato EDF (*European Data Format*), permitiendo que el usuario escoja el indicado; inmediatamente después, el programa determina si para este archivo existe el respectivo archivo con la configuración de los canales (archivo de montaje) y, si existe, lo carga haciendo la debida validación; si no existe, o es incorrecto, crea un montaje por defecto. Con el montaje, el programa determina las escalas a utilizar por cada señal; si es el caso, los respectivos filtros, y la posición de los electrodos en la visualización 2D (bloque ***Configura señales*** en el diagrama de la Figura 12). Seguidamente, el programa determina si existe un archivo válido de marcas para las señales y si existe lo trae a la memoria. Luego, examina para ver si existe un archivo de video; si existe, determina si el formato es soportado por la librería y, si lo es, crea la ventana de video respectiva. En este punto, en cada una de las ventanas activas se visualizan los datos correspondientes, y el programa está listo para procesar los eventos. A partir de aquí el

programa entra en un ciclo de procesamiento de mensajes y, cuando sea necesario, actualiza los objetos de visualización que sean modificados por dichos mensajes. De este ciclo solo se sale cuando se le indica ***Abrir*** otro archivo de señales (en cuyo caso repetirá los pasos a partir del conector 1), o cuando se escoge salir del programa.

El programa principal se encarga de la interface con el usuario, el acceso a los registros y la administración de los módulos. La Figura 13 muestra los componentes principales del programa principal del Visor.

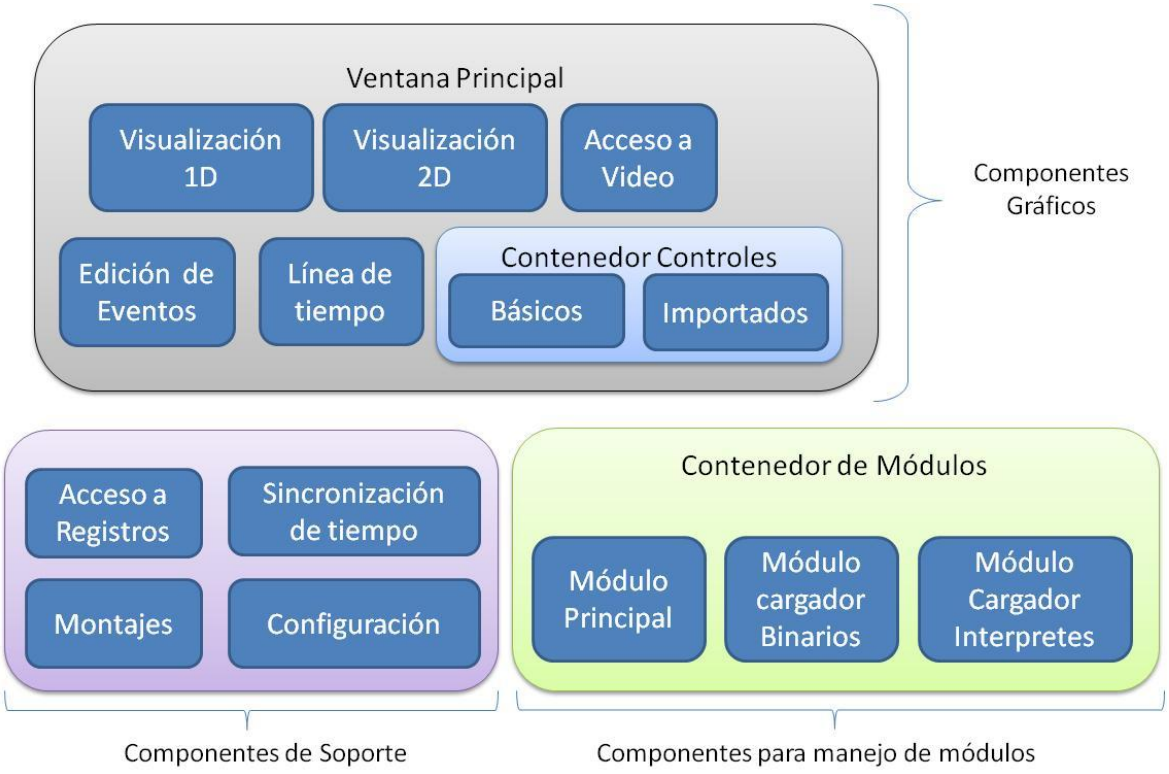


Figura 13. Componentes principales del Visor EEG_M

Los componentes aparecen agrupados según su función dentro del programa.

3.2 COMPONENTES GRÁFICOS

Los componentes gráficos son los que permiten desplegar la información de los registros: ya sea las curvas en la forma habitual de voltajes vs tiempo (Visualización 1D) o la representación topográfica (Visualización 2D). En esta categoría se presentan varios componentes:

- Un componente para la presentación del video correspondiente a un registro (este componente se desarrolló utilizando la librería *libvlc* de la organización *ViedoLan* que será descrita posteriormente).
- Otro componente para visualizar los eventos en una tabla que los despliega en formato texto. Aquí también se dispone de pestañas en las cuales se pueden configurar dichos eventos (cuales eventos visualizar, crear nuevos eventos, definir archivos de salida, etc);
- Igualmente se tiene un componente para mostrar la línea de tiempo donde se despliegan los eventos como líneas, y muestra la posición relativa de la visualización 1D o 2D con respecto a todo el registro.
- Por último se tiene un componente que se encarga de la presentación de los controles que permiten el llamado a algunas funciones del programa, ya sean estos los componentes originales, (que están compilados con la aplicación); o componentes que pueden adicionarse dinámicamente a partir de los módulos que se incorporen al programa.

Todos estos componentes están contenidos en la clase encargada de manejar la ventana principal del programa que se encarga de asignarles a cada uno de aquellos un espacio en pantalla y manejar los eventos más importantes del teclado y el mouse de la aplicación.

3.3 COMPONENTES DE SOPORTE

Todas las representaciones gráficas que se describieron en el apartado anterior deben sincronizarse para que al usuario siempre se le presente el mismo instante de tiempo en cualquiera de dichas representaciones; este componente se encuentra en la categoría de componentes de soporte. Aquí también hay un componente para el manejo de la configuración del programa, otro para todo lo que tiene que ver con la lectura de los registros en el archivo (este componente utiliza la librería Librasch que será descrita más adelante), y un componente para el manejo de los montajes de electrodos o de visualización de un archivo.

3.4 COMPONENTES PARA MANEJO DE MÓDULOS

El componente principal de esta categoría es el Contenedor de módulos, que es el encargado de explorar y cargar todos los módulos y/o intérpretes adicionados al programa (archivos ubicados en la carpeta modulos), y ejecutar las referencias a funciones de los módulos que son definidas en el programa principal y/o otros módulos, todo este proceso se explicará con más detalle en un apartado posterior. Como complemento a este componente se tiene otro componente encargado de definir los procedimientos y funciones compilados con la aplicación principal, otro para hacerlo con los módulos binarios (o archivos DLL), y un componente encargado de explorar y cargar todos intérpretes que se le adicionen al programa, así como los módulos adicionados en el lenguaje del respectivo intérprete.

3.5 DESCRIPCIÓN DETALLADA DE LOS COMPONENTES GRÁFICOS

Los componentes gráficos fueron desarrollados a partir de la librería *Visual Component Library (VCL) de Borland*, cuya estructura jerárquica aparece en la Figura 14.

La VCL se basa en el modelo de propiedades, métodos y eventos (PME). El modelo PME define los datos miembros (propiedades), las funciones que operan sobre los datos (métodos), y una manera de interactuar con los usuarios de la clase (eventos). La *VCL* es una jerarquía de objetos, escritos en Object Pascal e incorporados en el *IDE* de *C ++ Builder*, que permiten desarrollar aplicaciones rápidamente. Usando la paleta de componentes *C++ Builder* y el inspector de objetos, se pueden colocar componentes *VCL* sobre los formularios y especificar sus propiedades sin necesidad de escribir código [40].

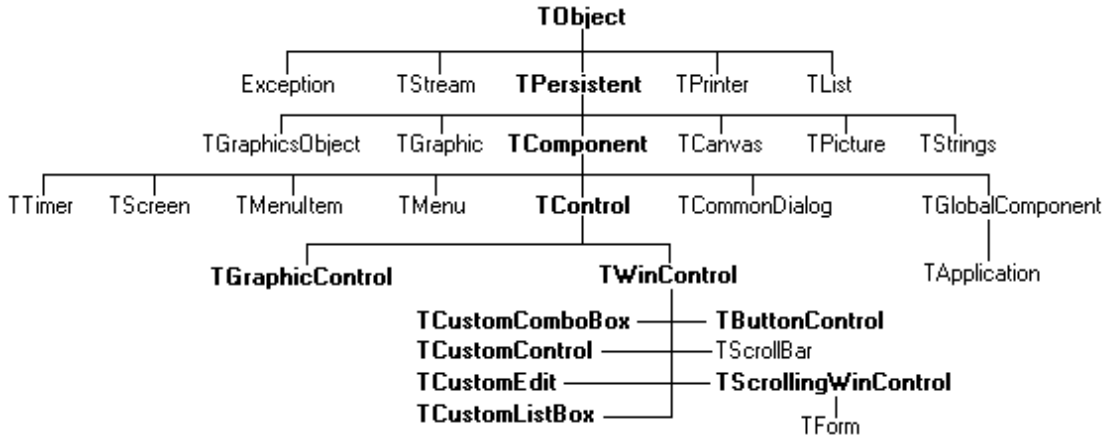


Figura 14. Estructura jerárquica de los componentes VCL.

3.5.1 Componente MainForm

Dentro de los componentes gráficos de programa, el más complejo es el objeto *MainForm*, una instancia de la clase *TMainForm* que como todos los formularios en VCL es una clase derivada de la clase *TForm* de la VCL, pues es el objeto que contiene a todos los demás.

En el constructor de esta clase se implementa la construcción e inicialización de los componentes principales del programa; para ello, se usan las siguientes funciones de inicialización:

void InitEventos(); *//inicializa los eventos*

void InitForms(); *//Inicializa los formularios del programa*

void InitModulos(); *//inicializa los contenedores de módulos*

void InitTimePos(); *//inicializa la posición en el tiempo*

void InitAccesoRegistros(); *//Inicializa clases para acceso a registros*

void InitToolBar(void); *//Inicializa la barra de herramientas*

void InitFrames(); *//Inicializa los paneles de la ventana principal*

void InitHelp(); *//inicializa el archivo de las ayudas y los respectivos tags*

// (marcadores)

Finalmente en el constructor se cargan los módulos instalables disponibles. Después de que se construye un Formulario, en la aplicación dispara el evento *OnCreate*, al cual se le ha asociado la función *FormCreate*, en esta función se lee el archivo de configuración a partir del cual se le dan las dimensiones a cada panel del programa para que aparezcan del tamaño en que se ajustaron la última vez que se utilizó el programa, esta configuración se guarda justo antes de finalizar el programa, cuando se dispara el evento *FormClose* de la clase. En esta clase se tienen algunas funciones y propiedades miembro que permiten la correcta sincronización entre todos los paneles, así como la ejecución de las funciones definidas en los módulos instalables. El principal objeto para la sincronización entre los módulos es el atributo *FileP*, que permite a todos los objetos que dependen del tiempo actualizarse cada que haya una modificación en la posición, o escala en el tiempo de visualización. También existe el componente *Registro* que maneja el acceso al registro de señales. Obviamente en el destructor de esta clase se borran todos los objetos creados en el constructor.

3.5.2 Componente Visualización 1D

Este componente se encarga de visualizar las señales en una gráfica voltaje vs tiempo de todos los canales que se dispongan en el montaje de visualización, cada uno con su respectiva escala en voltaje y todos con la misma escala en el tiempo, opcionalmente cada señal puede pasarse a través de filtros digitales pasa altos, pasa bajos o rechaza banda, con frecuencias de corte configurables por el usuario, también se puede quitar o poner divisiones en el tiempo (cuadrículas); el color de cada curva también hace parte de la configuración que se determina en el montaje. En esta gráfica adicionalmente aparecen textos indicando los eventos asociados a los registros que se marcaron en un momento dado. Por último cabe mencionar que en esta gráfica en algún momento dado el usuario puede marcar bloques para realizar algunas funciones.

Debido a que los registros de EEG regularmente contienen más de 20 señales, el diseño de este componente debe hacerse de tal manera que la representación no genere algunos efectos indeseables en la pantalla, como son el "parpadeo" o la demora en la actualización cuando se está haciendo un desplazamiento (*Scroll)* sobre la gráfica.

Para simplificar el uso de éste y los demás componentes gráficos se dividieron sus miembros en componentes lógicos y componentes visuales. Los componentes visuales, como su nombre lo indica son aquellos que hacen parte de la interfaz gráfica, mientras que los componentes lógicos son aquellos que permiten el acceso a los datos de los registros, la sincronización entre componentes y otras funciones indispensables para el correcto funcionamiento del programa. En este componente

se tiene que sus elementos visuales son aquellos que el usuario de la clase utiliza para indicar lógicamente los registros, la posicionan en el tiempo, el color, los filtros, las escalas a usar, la definición del modo del cursor, etc. (Estos son instancias de los componentes de soporte que aparecen en la Figura 13); y los componentes visuales, que son los que realmente "dibujan" la gráfica a partir de los parámetros indicados en los componentes lógicos. La Figura 15 muestra la relación de estos componentes.

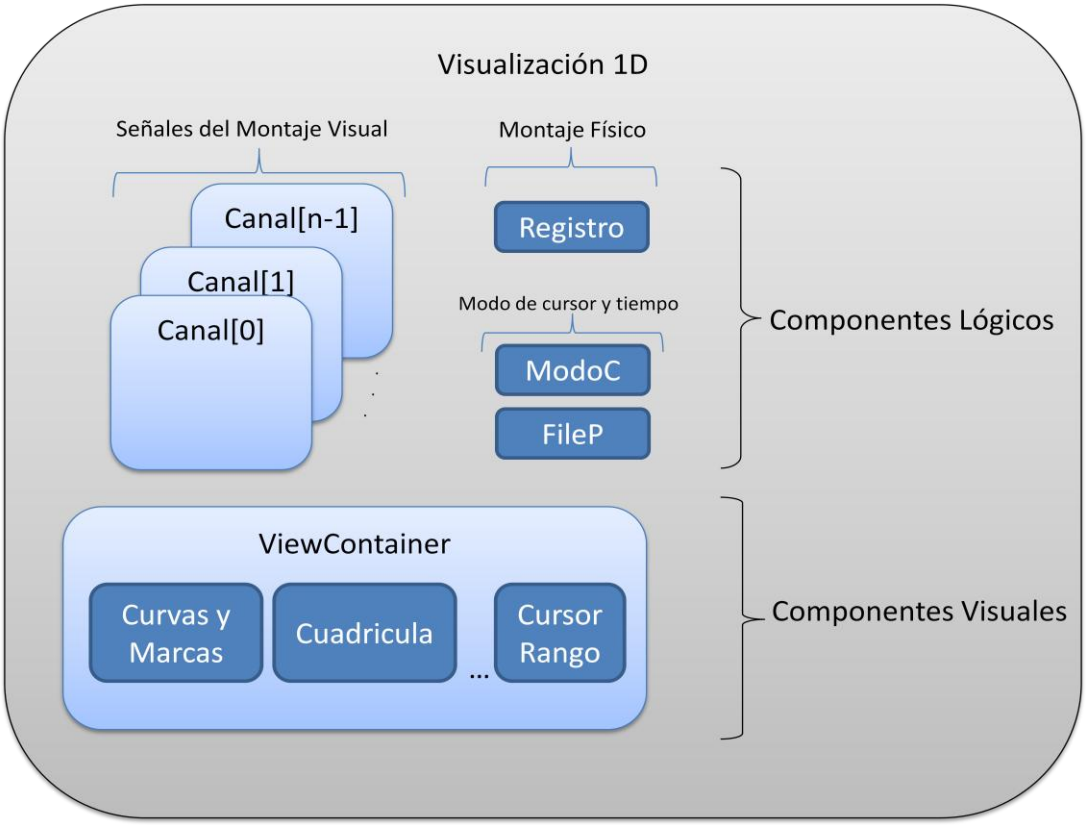


Figura 15. Componentes de Visualización 1D

3.5.2.1 Componentes lógicos

En la clase *TChannelBuff*, hay unos miembros que se encargan de definir la escala, el color, el tiempo, etc. de una señal, y unas funciones miembro que permiten acceder a los registros (utilizando la clase *rasch_access*) para obtener las muestras de voltaje en un instante de tiempo dado. La variable *Registro* contiene canales de tipo *TChannelBuff*, y son los canales tal como están contenidos en el archivo de señales (independientes del montaje visual). El arreglo *Canal* está contenido en la clase contenedora *TChannels_Group_Diff*, y cada canal es un objeto de la clase *TChannel_Diff* que es una clase derivada de la clase *TChannel_Buff*, que implementa su propia función *Get_Samples* (se leen los dos canales del registro y se hace una resta de las muestras en cada instante de tiempo). Para una descripción más detallada de estas clases consulte el Anexo 2. La variable ModoP es una instancia de la clase *TTiempo_Control* que se utiliza para mantener

sincronizados las diferentes visualizaciones del registro y se encarga de avisar cuando se deben actualizar las curvas por un cambio de posición en el tiempo o por un cambio en la escala de tiempo. De manera análoga se tiene el objeto ModoC que es una instancia de la clase *TModo_Cursor_Control* que se encarga de avisar cuando cambia el modo en que debe aparecer el cursor cuando pasa por la gráfica, qué hacer cuando se da click en la gráfica, etc.

3.5.2.2 Componentes Visuales

Aunque el Builder5 [41] tiene dentro de los componentes para agregar a los formularios un componente para dibujar curvas (denominado *TChar*), al realizar algunos ensayos se observó que este componente era demasiado lento para este propósito cuando se trazaban varias curvas, pues si bien no se generaba "parpadeo" de la pantalla, la actualización de la pantalla tardaba demasiado tiempo; por esta razón, se optó por crear una serie de componentes que cumplieran con esta función.

Para que la graficación no genere "parpadeo" de la pantalla se utiliza la técnica de doble buffer, en la cual se usa un buffer en memoria donde se hacen todos los cambios necesarios en un momento dado y una vez se tenga la gráfica definitiva se hace una copia del buffer a la memoria de pantalla [42]. Teniendo presente esta técnica se optó por tener una clase que contenga todos los objetos a dibujar, cada uno de los cuales se actualiza independientemente de los demás sobre la memoria que representa un mapa de bits (utilizando el componente *TBitmap* de las VCL) y cuando se tienen todos los objetos actualizados se juntan todos los mapas de bits en uno solo y de éste se hace copia del buffer de memoria al buffer de pantalla. Entonces se utilizaron tres mapas de bits, uno para la cuadrícula, otro para las señales y las marcas (la actualización de estos se hace simultáneamente), y uno más para el manejo de el sombreado en la selección de un rango (corresponden a los componentes *Cuadricula, Curvas y Marcas y CursorRango* respectivamente de la Figura 15).

El programa utiliza un componente contenedor denominado *ViewContainer* (instancia de la clase *TViewContainer*) , al cual se le adicionan todos los objetos a dibujar en la gráfica, y este a su vez contiene objetos que se derivan de una clase base denominada *TPanelView*, de la cual se heredan cada una de las clases encargadas de representar un objeto que debe actualizarse independientemente en la pantalla; algunos de ellos tienen asociado un mapa de bits y podrán contener objetos más elementales, que son los que realmente hacen algún "dibujo" en la pantalla. Estos últimos componentes hacen uso de un miembro de la clase *TBitmap* llamado *Canvas (*ambas clases hacen parte de la VCL*)*, por medio del cual se puede hacer el trazado de las curvas, poner texto, sombrear zonas, etc., utilizando funciones que acceden a las respectivas APIs de Windows. Por ejemplo para dibujar una señal se utiliza la función *Polyline* miembro de la clase *Canvas* para hacer polígonos, que a su vez utiliza la función *MoveTo* y *LineTo* de las APIs de Windows; las coordenadas de los vértices de estos polígonos se calculan a partir de los valores correspondientes a cada una de las señales (teniendo en cuenta los canales, la escalas de voltaje y tiempo respectivas y si es el caso después de haber sido filtradas).

3.5.3 Componente Visualización 2D

Este componente se encarga de visualizar en un mapa topográfico el valor de una función que se evalúa a partir de las señales de EEG en los puntos donde se ubican los electrodos y se interpola para los demás puntos de la corteza cerebral. La función a evaluar, así como el algoritmo a utilizar para interpolar los valores son definibles por el usuario (algunas de ellas se podrían incorporar a partir de los módulos instalables). También es posible escoger el tamaño de la cuadrícula que se aplica para la interpolación y la escala de colores a utilizar. La posición del electrodo respectivo a cada señal se determina de acuerdo al nombre de cada señal y a un archivo que contiene las posiciones de acuerdo al montaje físico que corresponda. En la Figura 16 aparecen los componentes que hacen parte de la Visualización 2d.

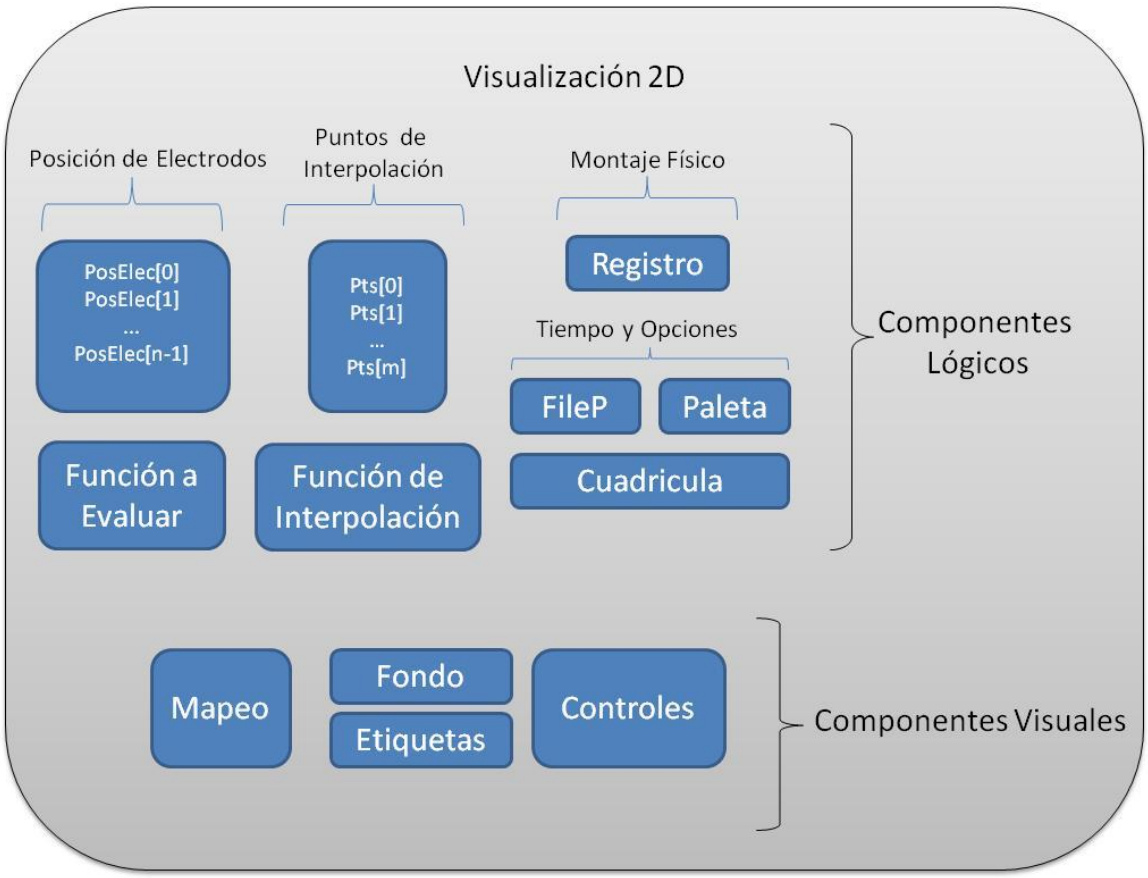


Figura 16. Componentes de Visualización 2D

3.5.3.1 Componentes lógicos

A partir del registro de señales se determinan los canales a graficar (en la Figura 16 el arreglo *PosElec*), y se determinan las posiciones de los electrodos ya sea a partir del archivo de posiciones que indique el archivo de configuración o, si éste no existe, buscando el archivo de posiciones que

más canales contenga de los indicados en el registro. De acuerdo al tamaño de la cuadrícula se crea un arreglo con todos los puntos de la imagen que deben ser interpolados (*Pts* en la Figura 16), teniendo en cuenta que deben estar contenidos en la imagen de la corteza. Seguidamente se invoca a la función a evaluar enviándole como parámetro la posición en el tiempo del registro (para esto se utiliza el componente *FilePos* de la Figura 16) y cada uno de los canales que hacen parte del montaje, tantas veces como canales haya, una vez se tienen los valores de la función para cada electrodo se invoca a la función de interpolación, para determinar el valor correspondiente para los puntos de la cuadrícula; finalmente, se invoca a la función que pinta cada retícula de la imagen con el color correspondiente a la paleta de colores escogida.

3.5.3.2 Componentes Visuales

El mapeo utiliza un objeto *TImage* de la clase VCL (*Visual Component Library*), cuya imagen original la toma de un archivo con la figura de una vista superior de la corteza cerebral (objeto *Fondo* de la Figura 16). De este objeto se toma el componente *Canvas* para acceder a los pixeles de la imagen para pintar cada región (su tamaño está determinado por el tamaño de la cuadrícula). Las etiquetas son objetos tipo *TLabel* cuya posición se calcula a partir del arreglo de posición de electrodos y su visibilidad se controla con un control *CheckBox.* La imagen de la cuadrícula simplemente se controla con otro control *CheckBox,* a partir del cual se dibujan o no las líneas en la imagen. Con un control *DrawGrid* se indica al usuario la escala de colores (Paleta).

3.5.4 Componente de Acceso al Video

Este componente tiene a nivel visual la apariencia habitual de un reproductor de video con un panel donde se visualiza la imagen, las típicas teclas de Play, Pause, Stop, además de un indicador del tiempo de reproducción correspondiente; la característica fundamental de este reproductor es que siempre está sincronizado con la de los visores de señales 1d y 2d del programa; para lograrlo se utiliza un miembro *FileP* que, como en las otros componentes, sirve para tal propósito. Este componente sólo se activa cuando al cargar el registro de señales el programa encuentra un archivo con el mismo nombre del archivo de registro, pero con extensión .MPG, o una vez cargado un registro se accede a la opción *Importar Video* del Submenú *Archivo*. La Figura 17 muestra los elementos principales de este componente.

Para el acceso al video primero se intentó con el componente *TMediaPlayer* de la clase VCL, pero no fue posible debido a que éste sólo reproduce sólo dos formatos de video (AVI y WEMF), dentro de los cuales no aparece el .MPG, que es el formato con el que se graba el video que acompaña a los registros obtenidos con el software ***CEEGraph***, que es el utilizado para obtener los registros en el Neurocentro. A raíz de esto, se realizó una búsqueda exhaustiva en Internet de componentes o librerías que cumplieran con tal propósito, y si bien se encontraron otras opciones para reproducir, al hacer los ensayos para posicionar la película en un instante de tiempo determinado (función esencial para poder sincronizar la visualización de los registros con el video), sólo la librería libvlc

lo hacía de manera satisfactoria, por esta razón y por ser de licencia GNU-GPL se optó por usar dicha librería.

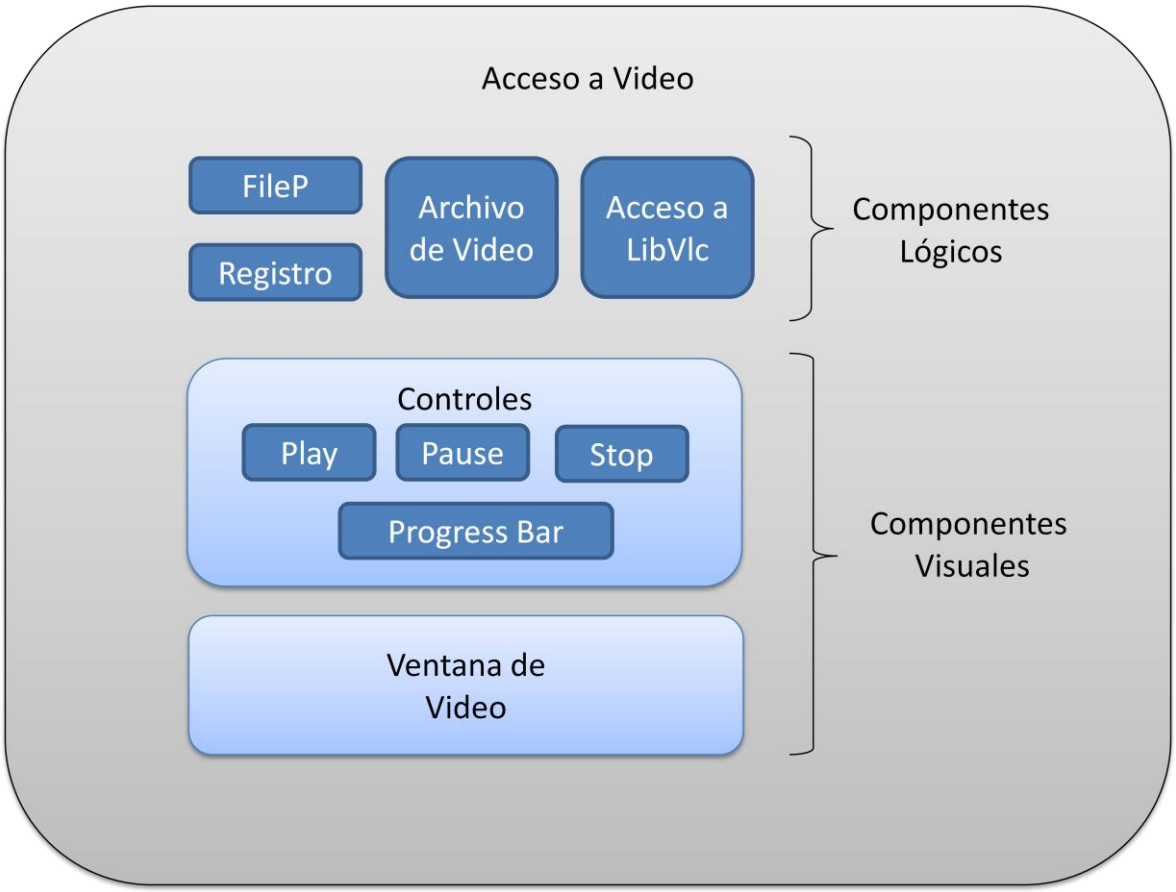


Figura 17. Componentes del Acceso a Video

3.5.4.1 Componentes Lógicos

Cuando se carga el programa, este componente busca el archivo *libvl.dll* (librería para el acceso al video del proyecto VideoLan), [43] y si lo encuentra lo carga, de lo contrario se desactiva el componente durante todo el programa. Luego, cuando se carga un *Registro* se busca el correspondiente archivo de video y luego invoca a la función *set_file_name* (si la función detecta un error, por ejemplo por el formato del archivo, se desactiva el componente), de la librería, si el manejador se asocia exitosamente se continua indicándole a la librería que el panel donde se va a mostrar es la *ventana de video*, para tal propósito se invoca a la función *set_Handle*, luego se invocan las funciones respectivas para reproducir, pausar o detener en el momento apropiado. Para mantener el tiempo de la película sincronizado con los demás componentes del sistema cuando el usuario utiliza las teclas de reproducción o cuando se hace un cambio en la posición en tiempo del registro, se utiliza *FileP*.

3.5.4.2 Componentes Visuales

La ventana es simplemente un objeto tipo *TPanel* (de VCL) del cual se toma el *Handle* para indicarle a la librería de video donde debe mostrar la película; el refresco y actualización de esta ventana está a cargo de dicha librería. Los botones son objetos tipo *TSpeedButton*, en cuyos eventos *OnClick* se ejecutan los llamados a las respectivas funciones de la librería, la barra de progreso es un objeto *TProgressBar*, cuya actualización se hace en la función de respuesta a un cambio en la posición del objeto *FileP* (en ésta función también se actualiza la posición de la pelicula), o cuando se está reproduciendo se utiliza el evento *OnTimer* de un timer (objeto de la clase *TTimer* de la VCL que permite realizar tareas periódicas), cuyo miembro *Interval* (indica cada cuanto se invoca a la función de respuesta al evento *OnTimer*), se ha fijado a 100 ms. A su vez en el evento *OnMouseDown* de la barra de progreso, se actualiza el objeto *FileP* para que todos los visores se ubiquen en una posición en el tiempo proporcional a la posición relativa del mouse, con respecto a dicha barra.

3.5.5 Componente de Línea de Tiempo

En este componente se visualiza un cuadro que representa todo el tiempo de registro y dentro de éste una barra que representa la posición y el tamaño relativo de la Visualización 1D, también aparecen en éste cuadro líneas que indican la ocurrencia de eventos en el registro (cuya configuración de colores y visibilidad se configuran en la ventana de edición de eventos); adicionalmente, aparece un cuadro indicando la posición y el tamaño del bloque de selección (en caso de estar en modo selección). A los lados de este cuadro aparecen botones de flechas que permiten avanzar y retroceder por páginas o por intervalos de páginas (el tamaño de la página y del intervalo se determina con la configuración en el tiempo del montaje). El usuario también puede dar *click* sobre cualquier posición del cuadro que representa todo el tiempo para ubicar las representaciones 1D y/o 2D en dicha posición relativa, y con el mouse en el modo selección se puede marcar el bloque. Los elementos que hacen parte de este componente aparecen en la Figura 18.

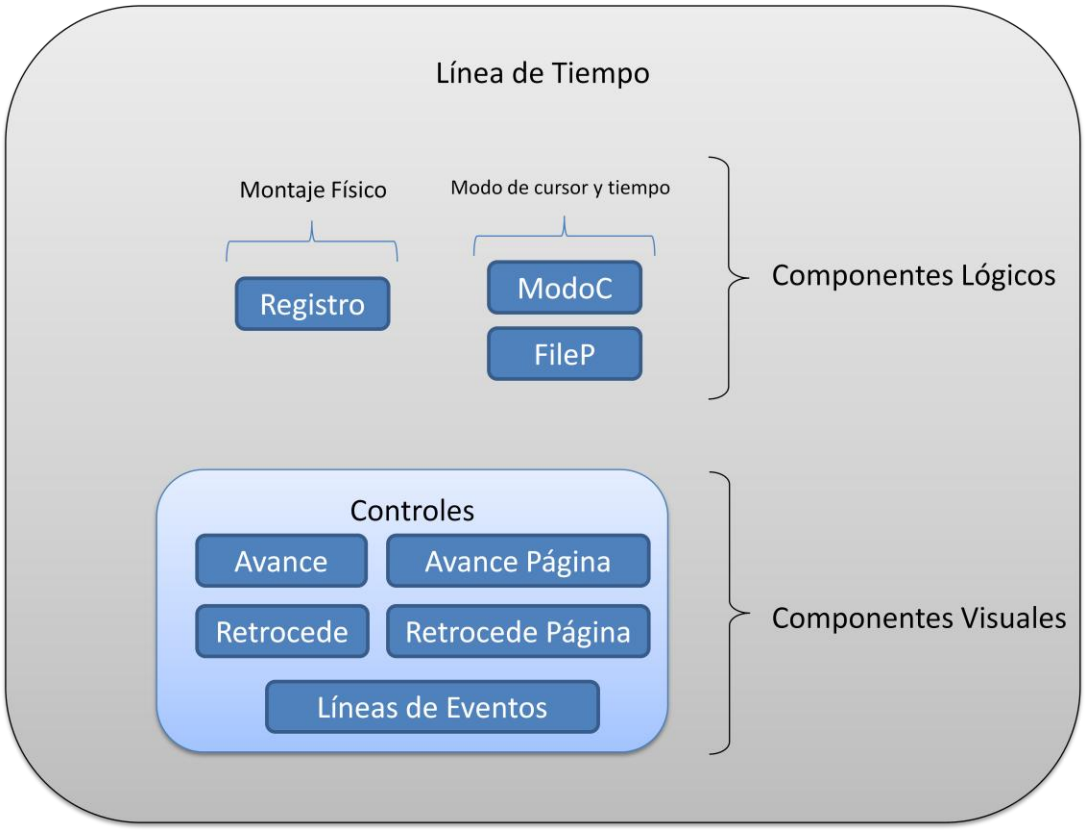


Figura 18. Componente de Línea de Tiempo

3.5.5.1 Componentes Lógicos

Cuando se carga el programa se determina el tiempo total del *Registro* para determinar el factor de escala entre el tamaño del cuadro que representa todo el tiempo, para usarlo en la representación de todos los elementos que allí aparecen; este proceso también se invoca cada que cambian las dimensiones de este componente. Con el miembro *FileP* se mantiene el sincronismo entre éste componente con los Visualizadores y el editor de Eventos; mientras que el componente *ModoC* se utiliza para indicar el modo del cursor, que puede ser: Normal, Selección y marcado de eventos. El componente *Registro* sirve también para determinar cuando ha habido un cambio en el montaje (en este caso lo referente a la configuración de los eventos), para hacer el refresco apropiado en la visualización.

3.5.5.2 Componentes Visuales

Los miembros *Avance, Avance Página, Retrocede y Retrocede Página* son objetos tipo *TButton* a los cuales se les define la función respuesta al evento *OnMouseDown*, que junto con la función *OnTimer* de un *Timer,* se encargan de hacer los cambios respectivos en el objeto *FileP* para que la

posición en el registro y en los visualizadores activos cambie de acuerdo a la escala de tiempo en una página o en una fracción de ella. El cuadro central que representa todo el tiempo es un objeto *TPaintBox* al cual se le definen las funciones de respuesta a eventos de la siguiente manera:

OnPaint: se encargan de dibujar las líneas de eventos, el cursor que indica la posición y el tamaño de la Visualización 1D, y el sombreado correspondiente al bloque de selección.

OnMouseDown, OnMouseMove: estas dos funciones son las que manejan los eventos del mouse sobre el cuadro para cambiar el tiempo en el objeto *FileP* y cuando se está en modo selección para que el usuario pueda marcar un bloque de selección en este cuadro.

3.5.6 Componente de Edición de Eventos

En este componente se maneja la información que tiene que ver con los eventos que se asocian al registro, ya sea que el marcado del evento lo realice el aparato, o los técnicos encargados en el momento de tomar el registro; o en otro momento, por el especialista (utilizando el programa CEGraph o el VisorEEG_M) o por alguna rutina de marcado que se le haya adicionado al programa. Con una pestaña se pueden ver los registros listados en orden de ocurrencia en el tiempo en un listado en el que aparecen: el tiempo en que se presentó el evento, el nombre del evento y el tiempo de inicio y final de dicho evento. En otra pestaña aparece un árbol con los tipos de eventos que aparecen en la pestaña anterior, ordenados en una estructura jerárquica, y en éste se indica si cada tipo de evento es visible y si lo es en qué color aparece (las líneas en el visualizador 1D y la línea de tiempo o el texto en la pestaña de eventos), también se puede asociar la tecla caliente que permite marcar un evento (cuando el programa se pone en modo marcado de eventos). En una última pestaña aparecen los archivos en los cuales se guardarán los eventos que se marquen por sesión del programa, también es posible activar o desactivar la presentación de eventos por archivo asociado. Los elementos que hacen parte de este componente aparecen en la Figura 19.

3.5.6.1 Componentes Lógicos

Cuando se carga un *Registro* el programa examina en la misma carpeta la existencia del archivo de eventos (un archivo con el mismo nombre del archivo de registro, pero con extensión .mrk), si este existe se lee su contenido para cargar el listado de todos los eventos teniendo en cuenta la configuración (el objeto *Conf_Ev*) de cuales eventos se quieren visualizar teniendo en cuenta las opciones dadas en el árbol de configuración y la herencia implícita entre los tipos de eventos. Cuando el usuario da *click* sobre un evento en particular se hace corresponder la variable tiempo del objeto *FileP*, para que los visualizadores se actualicen al tiempo en que ocurrió el evento. Recíprocamente, cuando ocurre un cambio en la posición del tiempo, la ventana de eventos se ajusta de tal manera que se puedan ver los eventos correspondientes a la porción de tiempo de la Visualización 1D. Cuando se está en el modo inserción de eventos, el usuario puede seleccionar cual evento agregar de la lista de eventos disponibles, (para agregar un evento simplemente se da *click* en la ventana de visualización 1D, en el instante de tiempo en que ocurre el evento).

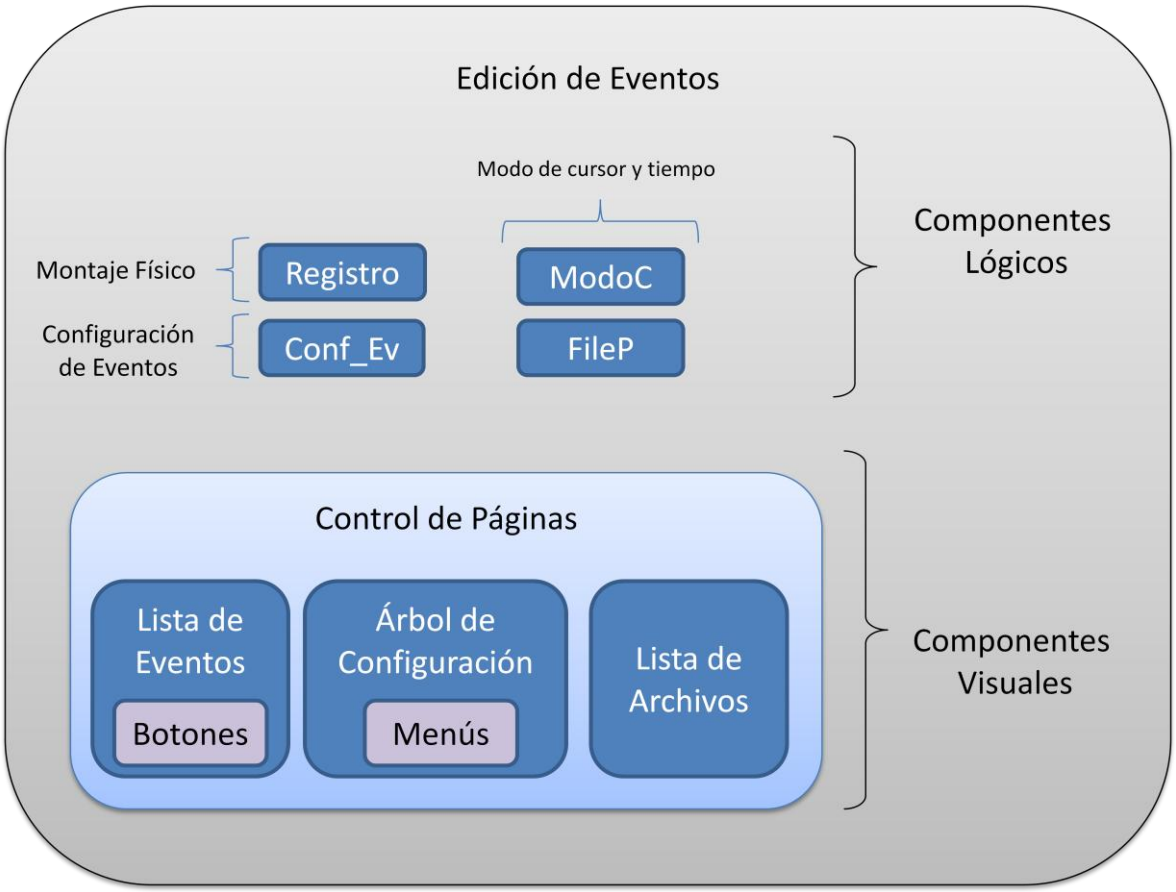


Figura 19. Componentes de Edición de Eventos

3.5.6.2 Componentes Visuales

El componente control de páginas es un objeto tipo *TPageControl*, y éste contiene un panel por cada una de las tres pestañas; el primer panel a su vez contiene un objeto *TDrawGrid* para la lista de eventos y los botones que permiten la selección, la inserción y el borrado de eventos. El segundo panel contiene un objeto *TTreeView*, que se encarga de mostrar la estructura de los tipos de eventos y al cual se le manejan los eventos *OnClick* para invocar los menús de configuración respectivos. El tercer panel contiene un objeto *TCheckListBox* que muestra todos los archivos que contienen eventos asociados a dichos registros y cuál de ellos se encuentra activo para asociar los nuevos registros a dicho archivo.

3.5.7 Componente Contenedor de Controles

Este componente contiene muchas de las funciones del programa a partir de componentes tipo caja de herramientas (ToolBox), cada una de las cuales tiene un comportamiento independiente y mediante un sencillo proceso de adición se incorporan a este panel. Estas cajas de herramientas

pueden ser parte del programa principal o pueden incorporarse mediante el proceso de agregar controles de los módulos instalables (proceso que será descrito más adelante). En el caso de los componentes propios de programa cada componente se elaboró por separado y se encapsuló en objetos tipo *TFrame*. Algunos de estos componentes poseen un miembro *FileP*, y/o un miembro *Registro* y/o un miembro *ModoC* para mantener el sincronismo con los Visualizadores. En la Figura 20 se muestran los miembros principales de este componente. Este componente se encarga de "repartir" el espacio entre las diferentes cajas de herramientas cada que cambian las dimensiones de la ventana principal y/o cada que el usuario desactiva o activa una de dichas cajas de herramientas (utilizando el control Control de Visibles); además, se comunica con la ventana principal para agregar opciones de menú que ejecuten las mismas funciones de cada caja de herramientas para que estén disponibles incluso cuando se oculte la vista de la barra de herramientas.

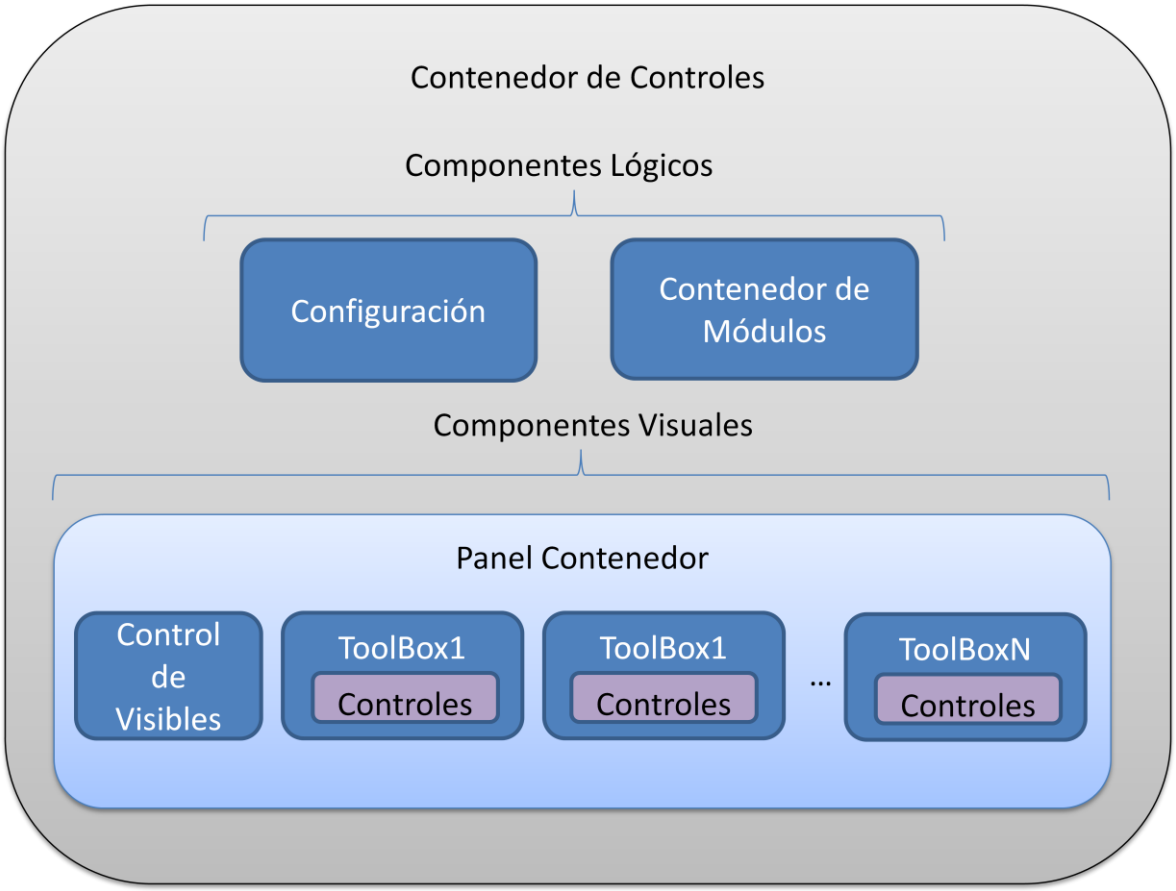


Figura 20. Componente Contenedor de controles

3.5.7.1 Componentes Lógicos

Cuando se inicia el programa se cargan en memoria todos los componentes que hacen parte de *Panel Contenedor*; inicialmente, los que hacen parte del programa principal y luego, los que hacen parte de los módulos instalables utilizando el objeto *Contenedor de Modulos*. Posteriormente, se

determina cuales cajas de herramientas (objetos *ToolBox* en la Figura 20, deben visualizarse partir del componente *Configuración*, el cual lee dicha información de un archivo XML. Simultáneamente se agregan títulos al menú del componente *Control de visibles*, que sirven para activar o desactivar cada caja de herramientas y se envía al programa los submenús apropiados para que aparezcan como parte del menú principal de la aplicación.

3.5.7.2 Componentes Visuales

El objeto Panel Contenedor es una instancia de la clase *TEEG_ToolBar*, la cual es derivada de la clase VCL *TPanel*, a la que se le han definido las funciones necesarias para que las cajas de herramientas se puedan agregar y/o posicionar apropiadamente cada que cambian las dimensiones de la pantalla y/o se active o desactive un componente de la barra de herramientas. Al crearse este componente se construye el objeto Control de Visibles, el cual es un objeto tipo *TSpeedButton* en cuya función *OnClick* se muestra un menú en el cual el usuario escoge cuales cajas de herramienta quiere visualizar o no, y éste le comunica a la barra de herramientas sobre la opción escogida por el usuario para ocultarla o visualizarla apropiadamente. Obviamente, cada que se agrega una caja de herramientas se le agrega una opción a dicho menú.

A continuación se explicarán las cajas de herramientas que vienen incorporadas con la aplicación, que se han denominado controles básicos (la estructura y la forma de construir los controles importados se detallará en secciones posteriores).

3.5.7.3 FilterFrame

Este control permite que el usuario escoja las diferentes configuraciones de filtro pasa bajos, pasa altos o filtro de ranura (filtro *notch*), para una o todas las señales que se seleccionen. Está compuesto por dos componentes tipo *TComboBox*, para los filtros pasa bajos y pasa altos y un componente tipo TCheckBox, para el filtro de ranura, además se tiene un objeto tipo Registro para comunicarse con la ventana principal. Cada que se cambia uno de estos componentes se hacen los cambios necesarios en el montaje a través de la variable Registro, y se da aviso a la ventana principal para que actualice el Visualizador 1D, y aplique los filtros en la forma indicada por el usuario.

3.5.7.4 EscalaFrame

Con este control el usuario puede elegir el tipo de cuadrícula (simple, doble o ninguna), la escala en el tiempo (en mm/s) para todas las señales, y la escala en Voltaje (en mV/mm) para una o varias señales. Además, el usuario puede escoger cuantas señales se verán simultáneamente en la ventana de Visualización 1D. Igual que el FilterFrame, este control se comunica con la ventana principal

con una variable Registro para que ésta le comunique al Visualizador 1D sobre los cambios hechos sobre el montaje, de tal manera que actualice apropiadamente la gráfica.

3.5.7.5 FrameModo

Con este control el usuario indica el modo del cursor que determina la forma como se comporta el programa cuando el usuario da *click* sobre el Panel de visualización 1D o la Línea de Tiempo.

3.5.7.6 FrameMontaje

Con este control el usuario puede elegir el montaje a utilizar para el registro cargado en pantalla, para lo cual el programa cada que el usuario da *click* sobre el botón respectivo, actualiza el archivo .XML que define el montaje visual del registro, luego se invoca al programa Montaje.exe, enviándole como parámetro el nombre del archivo mencionado, para que este programa se encargue de presentarle al usuario las opciones del montaje actual y las pueda modificar o, si lo prefiere, cambie todo el montaje de dicho registro. Al regresar de la ejecución de Montaje.exe en el programa se actualizan los cambios hechos en el montaje a partir del archivo XML, y se hacen los ajustes en la visualización que fueran necesarios.

3.6. DESCRIPCIÓN DE LOS COMPONENTES DE SOPORTE

En este grupo se encuentran una serie de componentes que sirven de apoyo al funcionamiento de los otros dos grupos (componentes gráficos y componentes para el manejo de módulos). El principal componente es el Registro, cuya función es permitir el acceso a las señales de un paciente en un momento dado. El par de clases *TTiempo_Atributo, TTiempoControl* se encargan de que cualquier cambio en la posición y/o rango de visualización sea actualizado en todos los objetos que así lo requieran. El componente Montaje permite definir las señales y sus características: sus escalas de visualización, sus filtros, sus colores, etc., se deben mostrar de acuerdo a la selección del montaje Visual y al respectivo montaje físico correspondiente a un registro. El componente de configuración permite llevar y traer hacia/desde un archivo XML, la información sobre la visualización (tamaño y posición de los paneles) y la configuración de los controles, y las asociaciones procedimiento-funciones que se definan durante la ejecución del programa. La descripción del funcionamiento de estos componentes aparece inmersa en la descripción de los otros grupos de componentes.

3.6.1 Componentes para el Manejo de Módulos

Con esta serie de componentes se hace la interface entre el programa principal y los módulos que se le pueden adicionar al programa. La Figura 21 muestra la relación entre estos componentes.

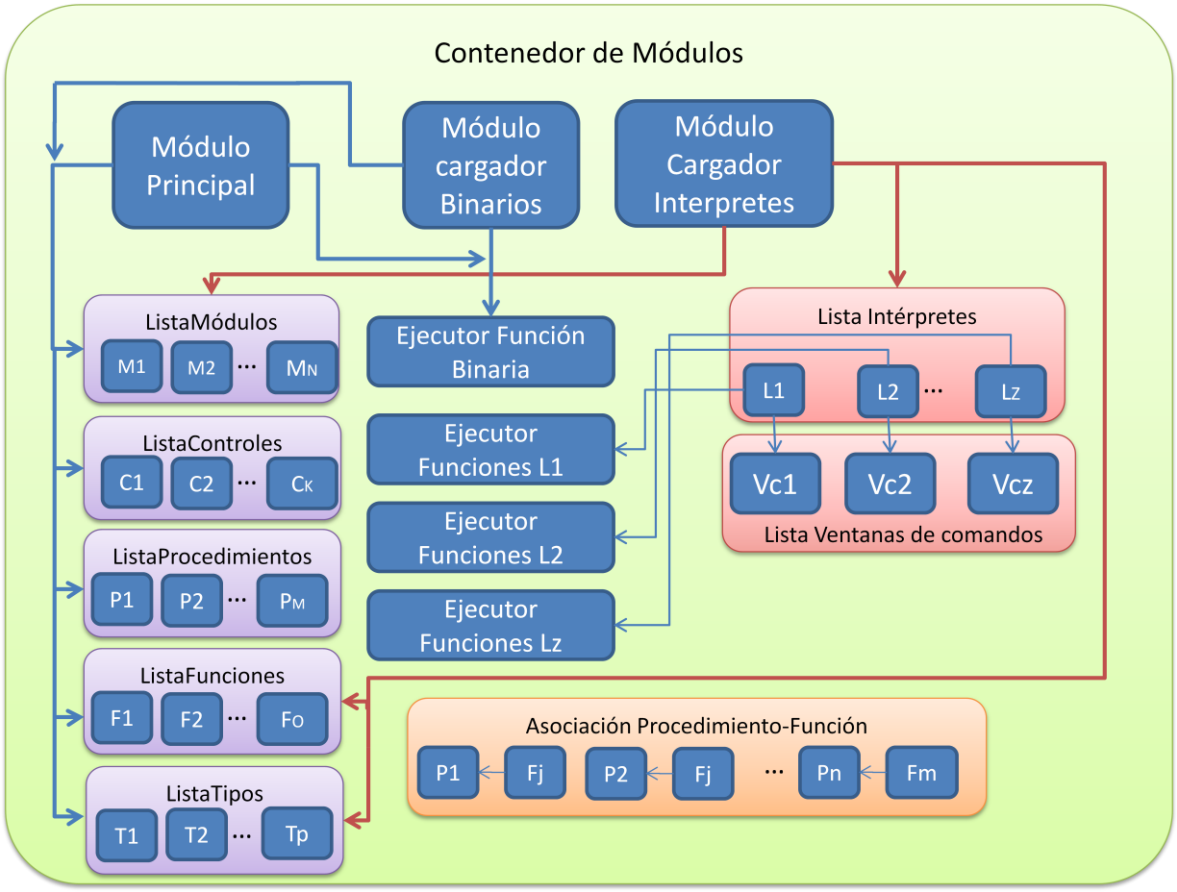


Figura 21. Componentes para el manejo de módulos

El código escrito en algún módulo está disponible para el usuario final mediante el uso de los *Controles*: declarando un control en dicho módulo y "exportándolo" al programa principal para que éste lo ubique dentro de la barra de herramientas del programa; la única restricción que se le pone a estos controles es que deben tener ciertas dimensiones para que puedan ser puestos en la barra de herramientas. En estos módulos también se pueden declarar *Funciones* para usarse en otros módulos, de tal manera que sea posible modificarse el comportamiento de los procedimientos que ya existen en el programa (el procedimiento puede ser declarado en la aplicación VisorEEG_M o en otro módulo), y a su vez en los módulos binarios también se pueden declarar *Procedimientos* para que puedan ser asociados a funciones que existan en otros módulos y/o en la aplicación VisorEEG_M (en el Apéndice 1 se describe el procedimiento para crear los módulos). Cuando se asocia una función a un procedimiento se hace una verificación de los tipos de datos de los parámetros y los valores de retorno de ambos y si no coinciden no se hace efectiva la asociación. Los tipos de datos para los parámetros y los valores de retorno de las funciones y procedimientos deben definirse mediante cadenas que identifiquen plenamente el tipo; si son tipos básicos del

lenguaje C (como *float, int ,char*) sólo basta con el nombre, pero si son tipos de datos más complejos como punteros o estructuras, primero debe registrarse el tipo de dato al contenedor de módulos (este lo agregará a la lista de tipos, en caso de que no exista y en caso contrario le informará si coincide o no con el tipo de dato que ya está registrado con ese nombre), esta operación puede hacerse en la aplicación principal o en un módulo adicional.

Los módulos binarios se compilan en forma independiente, como un archivo DLL (*Dinamic Link Library*) del sistema operativo Windows y deben ubicarse en la carpeta *modulos* que hace parte del sistema de archivos del VisorEEG_M. Para poder ejecutar los módulos interpretados se debe escribir un archivo de nombre *soporte.dll* que se ubica en una subcarpeta (cuyo nombre se asume que es el nombre del lenguaje que se interpreta), de dicha carpeta *modulos*; este archivo se encargará de servir de enlace entre el lenguaje interpretado y el VisorEEG_M, de tal manera que contenga las funciones para cargar, y ejecutar los módulos escritos en dicho lenguaje. Tanto para los módulos binarios como los módulos interpretados, los componentes aquí descritos sólo se encargan de cargar los módulos y ejecutar las funciones que se le solicite, otras especificaciones más complejas como crear tareas que se ejecuten en hilos o procesos independientes deben ser resueltas antes de invocar el llamado a la función.

Primero se describirá el procedimiento para cargar los módulos en memoria y luego el procedimiento para asociar funciones y procedimientos, y como ejecutar dichas funciones una vez estén asociadas a un procedimiento.

El componente contenedor de módulos se encarga de cargar los demás componentes en el momento en que se inicializa el programa. La aplicación crea este contenedor e inmediatamente le adiciona el componente módulo principal, y el componente módulo cargador de binarios. En el componente módulo principal se adicionan los procedimientos definidos en éste y también informa sobre las funciones que tiene la aplicación principal y que son públicas para que los demás módulos las puedan utilizar. Luego el cargador de módulos binarios se encarga de explorar en la carpeta *modulos,* cada uno de los archivos con extensión DLL, y para cada uno de ellos verifica que se trate de un módulo instalable; para serlo, el módulo debe contener como mínimo la función exportable la función *GetLibrary_Desc* (opcionalmente puede contener la función exportable *int InitLibrary(void)* que será invocada justo después de llenar las estructuras que definen el módulo). En caso de que el módulo no contenga dicha función, éste se descarta y se pasa al siguiente; después de esta verificación el cargador de módulos binarios agrega el módulo a la lista de módulos del contenedor, y luego invoca a la función *GetLibrary_Desc*, en la cual se espera que el módulo se encargue de llenar una estructura *plugin_infoDll,* que debe retornar con la cantidad de funciones, procedimientos y controles que tenga el módulo (además se deben indicar las asociaciones Procedimiento-Función por defecto y los conversores de tipo, que serán explicados más adelante), y los respectivos punteros; estas listas se adicionan a las respectivas listas de procedimientos, funciones y controles que contiene el módulo *Contenedor de Módulos*. La función *GetLibrary_Desc* tiene el siguiente prototipo:

plugin_infoDll GetLibrary_Desc(void*Handle ,EJECUTAFUNPROC Ejecutafunproc);*

El contenedor de objetos le pasa como parámetros a esta función un puntero a función y un puntero tipo void a un objeto tipo Module_Handle, para que dentro del módulo se puedan ejecutar los

procedimientos. Dependiendo de cómo se quieren invocar los procedimientos en el modulo usando solo Lenguaje C o usando C++, se hará uso de la función o del objeto (se usa un puntero tipo void al objeto para que los módulos escritos en solo C no generen error al no reconocer el tipo de dato). Este proceso se repite por cada archivo con extensión .DLL que se encuentre en la carpeta *modulos.* Inmediatamente después el objeto *Contenedor de Módulos* busca las subcarpetas contenidas en la carpeta *modulos*, y para cada una de ellas busca que contengan un archivo llamado *soporte.dll*; si no lo encuentra descarta esta carpeta, de lo contrario busca que esta librería contenga las funciones *Get_Module_Class* y *Free_Module_Class,* y si las posee agrega este módulo a su lista de intérpretes. Inmediatamente después, el intérprete adiciona su lista de funciones y la de tipos de datos (si existe), a las respectivas listas del Contenedor de módulos; se asume que el intérprete se encarga de explorar en la carpeta por cada uno de los archivos que pueden contener funciones y a partir de estos llena las estructuras que describen cada módulo y las que describen cada función. Un módulo interpretado puede ser un archivo o una carpeta (la decisión la toma quien hace el intérprete).

Todas las funciones que se adicionen al contenedor se invocan de la misma manera, un procedimiento contiene un índice de una lista de punteros al objeto función que se quiere ejecutar (si no se tiene ninguna función asociada contendrá un valor de -1), a su vez este objeto función contiene un valor que puede interpretarse de dos maneras: como puntero a una función, en caso de ser una función de un módulo binario; o como un índice de una lista de Scripts (archivo con código escrito en el lenguaje en particular) o módulo que lo contenga (el intérprete sabrá qué hacer con dicho valor), que debe ejecutarse en el caso de pertenecer a un módulo interpretado. El prototipo de estas funciones siempre es el mismo (más adelante se explicarán los detalles), y quien invoca dicha función se encarga de llenar los parámetros de forma apropiada (y cuando sea necesario, asignar la memoria dinámica), y luego le indica al contenedor que la ejecute; éste se encarga de determinar si la función es un Script interpretado o es código binario para llamar a la función *Ejecutor* apropiada, luego quien invoca la función se encarga de leer los valores de retorno de la función, y si es el caso destruir la memoria asignada para crear dichos valores de retorno y/o los parámetros a las funciones.

El intérprete, opcionalmente puede contener una función para mostrar una ventana de comandos para que el usuario invoque algunas funciones de prueba en el VisorEEEG_M, cada ventana se hará visible cuando el usuario escoja la respectiva opción en el menú *Modulos\Comandos* de la ventana principal. En el VisorEEG_M también se muestra una ventanas que sirve para determinar los módulos que están instalados en el programa, y revisar las funciones, controles y procedimientos, así como los tipos de datos que se utilizan en los módulos, el acceso a esta ventana se lleva a cabo utilizando la opción *Modulos\Modulos Instalados* del menú de la ventana principal.

En la estructura *plugin_infoDll* que contiene la información de cada módulo, opcionalmente se puede proporcionar las asociaciones Procedimiento-Función (en caso de que el módulo contenga procedimientos) por defecto que tienen los procedimientos de dicho módulo, para lo cual se utiliza la estructura *Proc_Funcion_Asociacion*, en la cual se debe pasar un puntero a la estructura *Procedimiento* y punteros a dos cadenas, una que indica el nombre del módulo que contiene la función y otra que indica el nombre de la función. Estas asociaciones se hacen efectivas (se verifica

la consistencia de los tipos y se busca el índice de la función respectiva) en el Contenedor de módulos una vez se cargan todos los módulos en memoria.

En la inicialización de la ventana principal se lee un archivo de configuración que contiene, entre otra información, las asociaciones *Procedimiento-Función*, y a partir de estas se toman los valores iniciales de los procedimientos, previa verificación de que dicha asociación pueda ocurrir (existencia de las funciones y módulos, coincidencia de tipos, etc.). En este archivo los procedimientos y funciones se guardan cada que termina la aplicación con el respectivo nombre (incluyendo el nombre del módulo al que pertenece), pues si se guardaran como índices, sus valores podrían diferir cuando se agrega o quita algún módulo.

3.6.1.1 Funciones y Procedimientos

Cualquier función que se incorpore como función "exportable" (es decir una función que se declare en un módulo para ejecutarse desde otro módulo utilizando la interface propuesta), deberá tener el siguiente prototipo:

int Nombre(void Parametros, void *Retorno);*

Donde *Parametros* apunta a un tipo de dato básico de lenguaje C o a una estructura que contiene los valores que se le envían a la función, de la misma manera *Retorno* apunta a tipo de dato básico de lenguaje C o a una estructura donde quedarán los valores que retorna en la función, el valor *int* que retorna se usa para determinar si se pudo evaluar o no la función (si retorna 0 se asume que la función se evalúo correctamente, de lo contrario indicará un tipo de error, y en la cadena cad_error del módulo, quien escribió la función debe explicar el error). Los tipos de puntero son void para que puedan adaptarse a cualquier situación. Así, tanto el usuario de la función, como quien la escribe deberán hacer las transformaciones de tipo necesarias en el momento de acceder a los datos. Se optó por este prototipo porque de esta forma se puede adaptar cualquier función, independiente de la cantidad de parámetros, e inclusive podrán codificarse funciones que "retornan" varios valores, pues simplemente se deberán definir apropiadamente las estructuras a las que apuntan *Parámetros* y *Retorno*.

Para establecer un mecanismo que garantice que una función se invoca con los parámetros, y que sean leídos los valores de retorno, se establece un mecanismo para indicar con cadenas los tipos de datos que hacen parte de cada uno y registrar dichas cadenas como "Tipos de datos". Dichas descripciones de los tipos de datos de los parámetros y/o valores de retorno simplemente son cadenas que representan el tipo de dato de cada miembro. Por ejemplo una función que requiera un parámetro entero y otro flotante tendrá asociada la cadena *"int float "*, y a esta cadena se le asocia un nombre que será el tipo de dato, por ejemplo *"datoIntF"*; este tipo de dato se debe registrar al contenedor de módulos, justo antes de que se use cualquier función que use el tipo en los parámetros o en el valor de retorno. Además, si existen intérpretes y se piensan usar funciones escritas en el lenguaje del intérprete que tengan parámetros o valores en el tipo registrado, también es necesario aportar en el intérprete o en módulos binarios el *conversor* de tipo, que no son más que

punteros a funciones para hacer que una variable de algún tipo en lenguaje C sea convertida a/desde su equivalente en el lenguaje interpretado.

Para tener una información detallada de cada función que se quiera exportar en algún módulo se debe crear una estructura tipo *función_def* para indicar los tipos de dato de los parámetros (campo *InputStr*), y los valores de retorno (campo *OutputStr*), además se deben aportar ciertos datos adicionales, como son: un nombre alternativo para el usuario (campo *UserName*), un texto descriptivo del propósito de la función (campo *Descripcion*), y el archivo de código fuente en el que está definido (campo *SourceCode*). Todos estos campos están contenidos en la estructura *DescripcionF* que a su vez hace parte de la estructura *funcion_def*.

Cuando se quiera definir una sección de código que será ejecutada en algún módulo (el programa principal se puede asumir como otro módulo), se debe utilizar una variable tipo *Procedimiento*. Este tipo de dato contiene un campo que es una estructura *DescripcionF* (es exactamente el mismo tipo que la estructura contenida en *función_def*), con la información que lo describe. Dicha información tiene dos propósitos, el primero y más importante es indicar el tipo de función que se puede asociar a él, para que el contenedor de módulos pueda validar una asociación en un momento dado, y el segundo propósito, es para que quien escribe los módulos sepa como implementar funciones que se puedan asociar a dicho procedimiento. Además de esta información un *Procedimiento* contiene un índice (campo *indexF*) que indica cual es la función que debe ejecutarse cuando se invoque.

Un procedimiento puede ser visible para el usuario o no (para lo cual se dispone del campo *UserCnf*), es decir que quien manipula el programa puede cambiar la función a la que apunta por medio de una interfaz que tiene el VisorEEG_M (la pestaña Configurar Procedimientos dentro del Panel de Procedimientos). Para este mismo propósito, a un procedimiento se pueden asociar dos funciones para comunicarse con el usuario, una para invocar un menú que configure algunas opciones del procedimiento y otra para mostrar los resultados después de invocar el procedimiento; además, éste contiene una lista de los índices de todas las posibles funciones que pueden ser invocadas por dicho procedimiento (para facilitar el cambio de función a la que apunta dentro del módulo que lo declara).

3.6.1.2 Registro de Tipos de Datos

Las cadenas que describen los parámetros y los valores de retorno de una función y el procedimiento asociado deben coincidir entre sí. Se recomienda que dichas cadenas sean nombres de tipos de datos registrados en el visor para que sea posible la verificación de tipos cuando se asocian funciones con procedimientos, además porque solo a los procedimientos que tienen como parámetros y valores de retorno tipos de dato registrados, luego es posible asociarle funciones en lenguajes interpretados utilizando los conversores.

Aunque registrar un tipo de datos y usar solo tipos de datos registrados, no son un requisito indispensable, constituyen un mecanismo que garantiza que no ocurran excepciones en el programa originadas porque el llamador de la función asume una estructura y en el cuerpo de la función se asume otra.

Cuando se carga un módulo se registran todos los tipos de datos que éste tiene y, si bien pueden existir definiciones previas con el mismo nombre, debido a que en otro módulo ya se hizo el respectivo registro del tipo de dato, si dichas definiciones son diferentes, el módulo que se está cargando será descartado; por esta razón, se deben tener presente los tipos ya instalados en el programa para que no ocurra tal conflicto.

Inmediatamente después de registrar los tipos de datos contenidos en cada módulo binario, se agregan los *conversores* de tipo que contenga el módulo para los intérpretes adicionados al programa. En cualquier caso, ya sea que el conversor lo ponga el intérprete o un módulo binario, se asume que los conversores para los parámetros y los valores de retorno existen en el momento de invocar a una función.

3.6.2 Ejecución de Procedimientos desde un Módulo Binario

Como se mencionó anteriormente, dentro de un módulo se puede invocar la ejecución de la función asociada a un procedimiento haciendo uso del puntero a objeto contenedor de módulos (tipo *ModuleContainer*), que se envía como parámetro a la función *Get_Library_Desc*, en caso de que el módulo esté escrito en *C++* (y haga uso del archivo de cabecera DllDriver.h). Esta opción solo está disponible para los módulos escritos en el compilador *C++ de Builder*, pues debido a que proceso de "decorado" de los identificadores para soportar la sobrecarga es propio de cada compilador [44]; si se utiliza otro compilador no reconocería las funciones miembro del objeto.

Pero si no está escrito en C++ (o no se utilice DllDriver.h), se puede hacer uso del segundo parámetro que se pasa al módulo cuando se invoca a *GetLibrary_Desc* y que es un puntero a función. El prototipo de este puntero a función es:

int(*)(void,Procedimiento*,void*,void*));*

Como primer parámetro se debe enviar el puntero *void* que se pasa a la función *Get_Library_Desc* (el puntero tipo *ModuleContainer*), el puntero al procedimiento que se quiere ejecutar, el puntero a la estructura que contiene los parámetros y el puntero a la estructura que contiene los valores de retorno.

3.6.3 Controles

Un control consiste en un objeto con interface de usuario que puede ser ubicado en uno de los paneles de controles que contiene el visor. El control puede ser creado utilizando objetos de la VCL en caso de estar compilado con *BorlandC Builder* o cualquier tipo de objeto de interface de usuario que defina el compilador respectivo. Los controles, igualmente a como ya se explicó para los otros elementos de un módulo binario, deben ser registrados en el proceso de carga de la librería por medio de un puntero a un arreglo contenido en la estructura *PlugIn_Info* que retorna la función *GetLibrary_Desc*. El funcionamiento del control, en cuanto a los eventos del mouse se refiere, será completa responsabilidad del módulo que lo contiene.

3.6.4 Interface con Otros Lenguajes (Intérpretes)

Para que al VisorEEG_M se le puedan adicionar funciones escritas en algún lenguaje interpretado es necesario disponer de un sistema que sirva de intérprete de dicho lenguaje. La principal tarea del intérprete es que tome un puntero a un objeto *function_def*, un puntero donde están los parámetros y un puntero donde están los valores de retorno y a partir de estos convierta los parámetros a variables de dicho lenguaje, ejecute el respectivo código y luego convierta los valores de retorno y los guarde en la posición de memoria indicada.

De la forma que se elija para interpretar el código escrito en algún lenguaje en particular sólo se exige que un listado de las funciones se cargue en memoria en el momento de instanciar el intérprete, y que se implementen las clases que se describen a continuación, de resto, quien desarrolle el intérprete es libre de escoger el sistema más apropiado.

Para implementar un intérprete para el sistema VisorEEG_M, se debe crear un archivo de enlace dinámico con el nombre *soporte.dll* (como ya se mencionó debe estar ubicado en una carpeta con el nombre del lenguaje que sea subcarpeta de *modulos*), que tenga como funciones exportables *Get_Module_Class* y *Free_Module_Class*. La primera de ellas deberá construir un objeto derivado de la clase *Module_Class*; a partir del puntero al objeto *Modules_Container* que se le pasa como parámetro y retornar 0 si lo pudo construir o de lo contrario retornar un valor distinto de 0 (en cuyo caso será removido de la lista de intérpretes). La segunda función será invocada cuando se quiera eliminar el intérprete de la memoria.

Opcionalmente el archivo puede exportar la función *Show_Commands*, que se utiliza para mostrar una ventana donde el usuario puede ejecutar comandos en dicho lenguaje. El objeto instanciado de la clase derivada de *Module_Class*, sólo se usa de puente entre el Contenedor de módulos y los módulos del lenguaje, mediante la función *Find_Modules*, que se utiliza para buscar todos los módulos que se incorporen al Visor en el lenguaje específico. A esta función se le envía una trayectoria y en ella se debe crear la lista de módulos, y por cada módulo agregar las respectivas funciones que pueden ser agregadas al visor. Qué es un módulo y cómo determinar que una función que hace parte de un módulo puede agregarse a la lista de funciones del Visor, depende exclusivamente de la implementación de la clase en cuestión. Por ejemplo, en *Matlab*® y en muchos lenguajes interpretados una función puede escribirse en un archivo por separado, entonces una idea simple es que cada módulo se construya con base en una carpeta y cada función con base en un archivo (de hecho esta fue la manera como se implementó el intérprete de Matlab®, que más adelante se explica con más detalle).

Cada módulo creado por el intérprete debe ser una instancia de una clase derivada de *Module_Handle*, con su propia versión de la función *Exec_Function*, que será invocada en el momento en que el Contenedor de Módulos detecte que a un procedimiento se le asocia una función escrita en dicho lenguaje; a dicha función se le pasará un puntero a un objeto tipo *function_def*, en cuyo campo *m_index* se le envía el índice que el contenedor de módulos respectivo puso para identificar a la función. Además se le pasará el puntero a donde se encuentran los parámetros y otro en donde deben quedar escritos los valores de retorno.

Lógicamente el primer paso que deberá hacer el objeto es crear a partir de la dirección de los parámetros y la dirección del valor de retorno, las respectivas variables en el lenguaje interpretado; el segundo paso, será ejecutar la función; y finalmente, a partir de la variable que contenga el valor de retorno en el lenguaje, guardar los valores en la dirección donde se contiene el valor de retorno. Para este proceso se recomienda el uso de los Conversores. Igualmente a como ocurre con las funciones creadas en módulos binarios, la función *Exec_Function* deberá retornar 0 si se pudo ejecutar la función y cualquier otro valor en caso contrario (además puede poner un texto explicativo del error en la variable *cad_error* del módulo).

3.6.4.1 Intérprete de MATLAB®

MATLAB® es un poderoso lenguaje de programación que incluye los conceptos comunes a la mayoría de los lenguajes de programación. Puesto que se trata de un lenguaje con base en scripts, la creación de programas y su depuración en MATLAB® con frecuencia es más fácil que en los lenguajes de programación tradicionales como C++ [45]. Por ser Matlab® es uno de los lenguajes más utilizados para implementar algoritmos de procesamiento de señales [46] se decidió hacer el respectivo intérprete para el Visor.

El intérprete se hizo de tal forma que se pueda adaptar cualquier función hecha en Matlab® para que sea utilizada en el Visor. Para cumplir con este objetivo se utilizó una serie de bibliotecas de enlace dinámico que vienen con Matlab®, diseñadas especialmente para ejecutar código escrito en dicho lenguaje desde un programa ejecutable (conocidas como *Matlab® Engine*). Las funciones contenidas en estas librerías (DLL), se pueden utilizar gracias a una serie de archivos .LIB que la empresa MathWork distribuye con la aplicación de Matlab®. Estos archivos por su propia naturaleza son dependientes del sistema operativo y del compilador que se use para escribir el ejecutable o la librería de enlace dinámico. Desafortunadamente, a partir de la versión 8.5 de Matlab®, el compilador de Matlab® solo incluye soporte para el compilador propio lcc y para el Visual C++ [47] lo que significa que no se incluyen los archivos .LIB para los desarrolladores que utilizan el compilador del BorlandC Builder 5, que como ya se mencionó, es el compilador que se escogió para el desarrollo del Visor. Por esta razón fue necesario escribir parte de este intérprete en el compilador Microsoft Visual C++ (versión 7), en un módulo .DLL que accede a las funciones del Engine de Matlab®, y que tiene funciones exportables en lenguaje C, que pueden ser invocadas desde el módulo principal *soporte.DLL* hecho en BorlandC Builder. A continuación se describe como se construyó éste módulo y los otros componentes del intérprete.

Para implementar el intérprete de Matlab® se generó un archivo *soporte.DLL* y se ubicó en la subcarpeta *modulos\Matlab®* del Visor. En este archivo, además de las funciones *Get_ModuleClass* y *FreeModuleClass*, se implementa la clase *Module_ClassMatlab®* derivada de la clase *Module_Class* (de la cual se instancia el objeto construido y destruido con las dos funciones mencionadas). El objeto mencionado se encarga de explorar las carpetas contenidas en *modulos\Matlab®* en busca de subdirectorios y por cada uno de ellos crea un objeto tipo *Module_Matlab®*, y adiciona al path de Matlab® dicha carpeta. La clase *Module_Matlab®* derivada de la clase *Module_Handle*, sirve para implementar la respectiva función que ejecuta un

Script de Matlab® dentro del visor, así como la conversión de variables de C++ a variables de Matlab®. La clase Module_ClassMatlab® utiliza una serie de funciones definidas en otro módulo DLL (*EngModVC3.DLL*) compilado en Microsoft VisualC++ en donde se escribieron las funciones que acceden directamente a las API de Matlab® para invocar Scripts desde un ejecutable, a partir de las cuales se instancia un objeto tipo *Matlab®_Engine* para encapsularlas en una clase. A continuación se presenta una breve descripción de dichas funciones.

Para invocar la ejecución de las funciones de la API de Matlab® desde un módulo externo (ya sea un ejecutable o un archivo DLL), se debe crear una sesión del Engine de Matlab® utilizando la función *engOpen*, que retorna un puntero a un manejador que se usa como referencia para la ejecución de cualquiera de las demás funciones. Para la ejecución de código en Matlab® se invoca a la función *engEvalString* que toma una cadena y evalúa la expresión contenida en ella. La forma de establecer las variables comunes entre un programa ejecutable y el Matlab® es por medio de variables tipo mxArray, que permiten definir cualquiera de los tipos propios o creados en Matlab®. Para la conversión de variables de mxArray en C a variables propias del engine de Matlab® y viceversa se utilizan las funciones *engGetVariable* y *engPutVariable* respectivamente, más una serie de funciones para crear los mxArray a partir de variables simples en C (*mxSetDimension, mxSetData, mxGetDimension, mxGetData, mxSetFieldByNumber, etc*).

Al finalizar la sesión se debe usar la función *engClose*. Para una descripción completa de estas funciones consultar [47] La invocación a estas funciones se escribieron en el archivo *EngModVC3.DLL* que se comunica con un objeto tipo ***Module_ClassMatlab®*** de *soporte.dll* por medio de funciones en C exportables y cuyas declaraciones están en el archivo ***engmod.h***. Para una revisión más detallada del funcionamiento del intérprete de Matlab® se debe consultar el Anexo 1.

3.6.4.2 Módulos de MATLAB®

Los módulos en Matlab® para el Visor se implementaron como simples carpetas contenidas en el subdirectorio *modulos\Matlab®* que contienen las funciones propias del módulo en un script por cada función y un archivo texto que lo describe. A continuación se hará una breve descripción de cómo se debe escribir la función y como es el archivo texto que describe al módulo.

La sintaxis de Matlab® simplifica el uso de variables pues no requiere que estas sean declaradas, en particular las variables que se utilizan como parámetros o como valores de retorno. Esto dificulta la interface con el Visor, pues como ya se mencionó por cada función a incorporarse en un módulo debe pasarse una estructura que contenga los tipos de datos de los parámetros y los valores de retorno. Para resolver este problema se optó por la siguiente solución:

Para que el visor ejecute una función escrita en Matlab® dentro de un procedimiento definido en algún módulo ésta debe escribirse de tal forma que pueda establecerse la cantidad y tipo de parámetros y valores de retorno en lenguaje C a partir del mismo archivo que la contiene (para poder hacer la respectiva conversión entre el *engine* y el programa). Para este objetivo se estableció que dicha información se definiera como comentarios que se deben adicionar al script de la función inmediatamente después de la línea que la declara. Adicionalmente en estos comentarios se debe

escribir una breve descripción de la función. Entonces toda función debe iniciar con el siguiente encabezado:

*function [<**Lista de valores de retorno**>] = <NombreFuncion>(<**Lista de parámetros**>)*

*% < **Nombre Funcion para Usiario**>*

*% <**Descripcion**>*

*% Parametros:<**Nombre tipo en Lenguaje C**>*

*% Retorno:<**Nombre tipo en Lenguaje C**>*

Se asume que se pone un solo nombre en *<Nombre tipo en Lenguaje C>* para los parámetros o para el retorno y si la función tiene más de un parámetro o más de un valor de retorno, el nombre debe de corresponder con una estructura en lenguaje C que contenga todos los parámetros o todos los valores de retorno respectivamente.

El archivo que contiene la función debe estar ubicado en una carpeta que sea subcarpeta de *modulos\Matlab®*, del subdirectorio de la aplicación. En esta misma carpeta debe ubicarse un archivo con nombre d*escripcion.txt*, que debe seguir el prototipo que aparece a continuación:

*Autor: <**Nombre del Autor del Módulo**>*

*Descripcion:<**Descripción de las Rutinas contenidas en la carpeta**>*

*Version**:<Version**>***

*Fecha:<**dd.mm.aaaa Fecha de creación**>*

3.6.4.3 Conversión de Variables de Lenguaje C a MATLAB®

Si el nombre del tipo en lenguaje C no es un tipo de dato básico de lenguaje C (por ejemplo si hay más de un parámetro o más de un valor de retorno), se debe proporcionar en un módulo .DLL. El intérprete ya posee los conversores para los tipos de datos básicos y algunas estructuras que se consideraron podrían ser útiles (sus prototipos están en el archivo engmod.h). Esto se hace declarando una estructura tipo *mxConvert* en cuyos campos debe contener el nombre del tipo y punteros a funciones para crear, copiar y liberar memoria durante el proceso de conversión entre una variable en C de dicho tipo y la respectiva variable mxArray y viceversa, para comunicarse con el engine de Matlab® (en estas funciones lo más probable es que tengan que usarse las funciones de la API de Matlab®, y por lo tanto en el archivo DLL que se construya para tal propósito se debería incluir en archivo EngModVC3.LIB). El prototipo de esta estructura y sus campos que también aparece en el archivo *engmod.h*, es el siguiente:

```
struct mxConvert
{
 char *nameTipo;
 mxArray* (*Create_mxArray)(void *,char *);
 void*  (*Create_CVar)(mxArray*,char*);
 int (*Set_mxArray)(mxArray*,void *,char *);
 int (*Set_CVar)(mxArray*,void *,char *);
 int (*Free_mxArray)(mxArray*,char *);
 int (*Free_CVar)(void*,char *);
};
```

Donde *nameTipo* es el nombre del tipo de dato registrado en lenguaje C, y los demás campos son punteros a funciones que se encargan de crear el espacio en cada ámbito (*lenguaje C* o *Matlab*®), de copiar de un lenguaje a otro, o de liberar la memoria en cada ámbito. Cuando es un solo parámetro o un solo valor de retorno, simplemente estas funciones utilizan la estructura *mxArray* a la cual se le asigna el valor respectivo a partir de la memoria donde se encuentra la variable en lenguaje C (utilizando las funciones respectivas que están declaradas en *engmod.h*). Si existe más de una variable en los parámetros o en los valores de retorno, la variable *mxArray* deberá ser una estructura en la que cada campo deberá corresponder con una de las variables que hace parte de los parámetros (o los valores de retorno si se están declarando para el retorno).

3.6.4.4 Ventana de Comandos de MATLAB®

Al intérprete de *Matlab*® se le adicionó la función *ShowCommand* para que el usuario pueda invocar comandos de *Matlab*® directamente en el visor para hacer algunos ensayos como si estuviera ejecutando el programa *Matlab*® y de esta manera depurar fácilmente las funciones escritas en dicho lenguaje. Como ya se mencionó a esta función se accede en un submenú de la opción *Modulos-Comandos* del programa principal. Esta función muestra el cuadro de diálogos que aparece en la Figura 22.

En la pestaña *Entrada* el usuario digita los comandos como si estuviera en la ventana de comandos de *Matlab*®. En la pestaña *Salida* se muestra el resultado de evaluar las expresiones digitadas. En la pestaña *History* se muestra un historial de todas las expresiones evaluadas y en la pestaña *Variables* se muestran todas las variables utilizadas en el *Engine.*

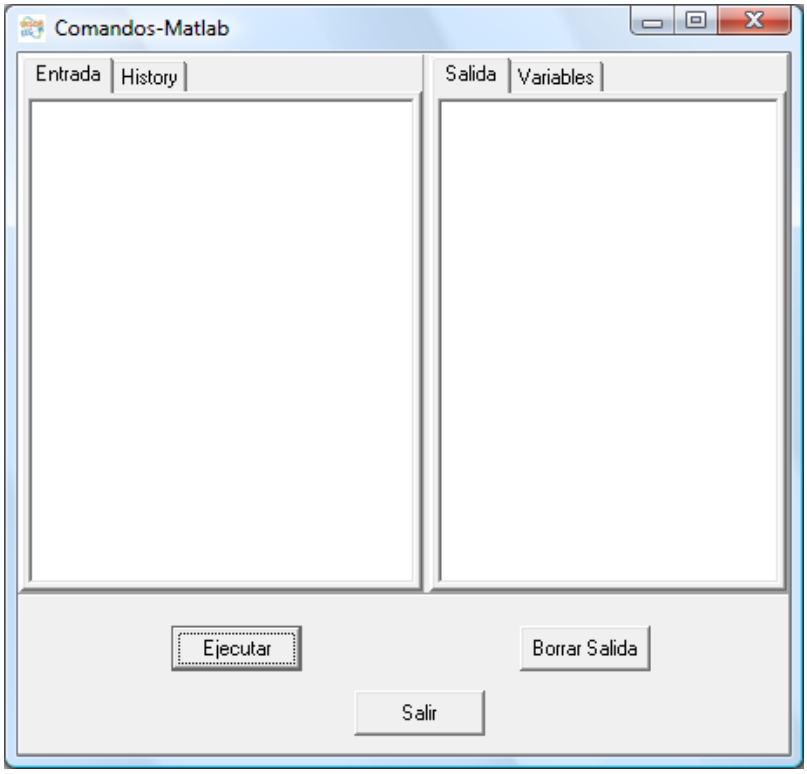


Figura 22. Ventana de Comandos de *Matlab*®

4. INSTALACIÓN Y MANEJO BÁSICO DEL VISOR EEG_M

En este capítulo se dará una breve descripción del manejo del programa desarrollado, para una información más detallada consultar los Anexos 3 (Proceso de instalación del programa), 4 (Guía del usuario) y 5 (CD con el programa y las ayudas del *Visor_EEGM*).

4.1 REQUERIMIENTOS DE SOFTWARE PARA LA INSTALACIÓN DEL VISOR EEG_M

El programa *Visor_EEG_M* es una aplicación que funciona bajo el sistema operativo Windows, en las versiones iguales o superiores a Windows XP, con procesador Pentium II (450MHz) o superior, con una memoria RAM disponible de 64 MBytes y ocupa un espacio de 400 MBytes de disco duro.

Para el funcionamiento básico de visor deben instalarse las librerías ***Librasch 4.5***, y el programa *VideoLAN*, ambos de distribución gratuita bajo licencia GNU y contenidas en el CD de instalación (en el programa instalador, éstas se instalan automáticamente cuando se selecciona instalación completa).

Si el usuario tiene instalado *Matlab*® versión 7 o superior, el programa podrá ejecutar *Scripts* de *Matlab*® como módulos que se incorporan al programa.

4.2 PROCESO DE INSTALACIÓN

Una vez insertado el CD, el programa de instalación inicia automáticamente, si esto no ocurre abrir la carpeta de la unidad y ejecutar el programa setup.exe. En la pantalla aparecerá un menú para ir instalando cada una de las aplicaciones o instalar todo. Para más detalles del proceso de instalación consultar el Anexo 3.

4.3 DESCRIPCIÓN BÁSICA DEL VISOR EEG_M

En la Figura 23 se muestra las diferentes partes que conforman la interfaz con el usuario del programa *VisorEEG_M*, en ella podemos identificar siete paneles diferentes cada uno de los cuales tiene un propósito definido:

Panel 1. En este panel se pueden ver los registros que hacen parte del montaje. Todos con la misma escala de tiempo pero cada uno puede tener una escala en voltaje diferente, así como la opción de aplicar o no filtros digitales pasa-bajos, pasa altos y/o rechaza banda.

Panel 2. En este se puede visualizar una representación topográfica de los niveles de voltaje en las diferentes zonas de la corteza cerebral calculados a partir de los correspondientes a las señales del

registro. El tamaño de la cuadrícula, la función a calcular en la representación, así como la escala de colores son configurable por el usuario.

Panel 3. En este se puede ver una serie de controles que permiten la aplicación de los diferentes algoritmos que están disponibles para aplicar a los registros a partir de los módulos binarios o interpretados que están disponibles para el usuario.

Panel 4. Aquí se visualiza el video correspondiente al registro de las señales; éste se encuentra sincronizado con las otras visualizaciones para poder observar el comportamiento del paciente en un instante de tiempo, simultáneamente con las representaciones 1D y 2D.

Panel 5. Ventana que permite mostrar las marcaciones hechas en el momento de la obtención del registro y/o marcaciones posteriores hechas por el especialista o por rutinas implementadas en módulos instalados en el programa.

Panel 6. Controles que permiten visualizar y controlar la posición en el tiempo de la porción de registro visualizada en los paneles de visualización.

Panel 7. Controles que permiten la configuración de las escalas de tiempo, la aplicación de filtros, y los controles instalados en los módulos agregados al programa.

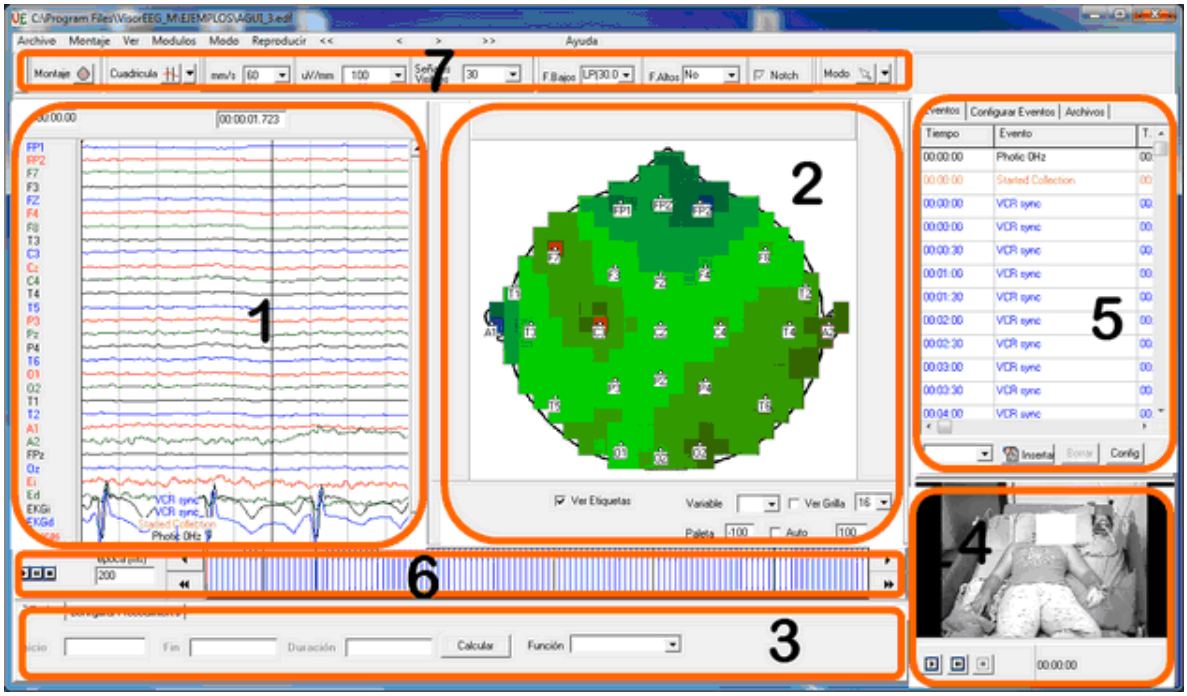


Figura 23. Interfaz con el usuario del programa *VisorEEG_M*

Para una descripción más detallada del manejo del programa consultar la Guía del Usuario que aparece en el Anexo 4.

5. RESULTADOS

Dentro de los objetivos propuestos todos ellos se lograron llevar a cabo en un gran porcentaje. A continuación se mencionaran los más relevantes:

Después de una búsqueda exhaustiva de las posibles librerías para acceder a los archivos de los registros EEG se optó por utilizar la librería *libRASCH*. Esta se adaptó completamente al entorno *C++ Builder*, pues originalmente viene para el entorno Visual Studio. A partir de las rutinas desarrolladas basadas en esta librería se puede acceder a los registros independientemente de la frecuencia de muestreo y del formato en el que está almacenado (se pueden usar los formatos manejados por la librería *libRASCH*). También se logró acceder a las marcas de los eventos que genera el software *CeeGraph* en el momento de la toma de los registros y/o en análisis posteriores, con rutinas propias e inclusive se permite generar marcas nuevas. En el programa, el usuario puede ver la zona del registro donde se ubicaron las respectivas marcas.

Se logró diseñar un grupo de rutinas de visualización propias, que acceden a un nivel más bajo que los componentes VCL a las API de Windows y, gracias a su desempeño, se pueden observar en diferentes escalas de tiempo y niveles de voltaje cualquier cantidad de señales del registro EEG, sin que se observen efectos indeseados en la pantalla.

Se implementaron rutinas que permiten acceder a los *frames* de video del archivo asociado a un registro de EEG, utilizando la librería *libvlc* de VideoLan, para un mejor correlación de las señales EEG con el comportamiento del paciente durante el examen. En la Figura 24, puede observarse el momento en que se presenta un episodio epiléptico: el programa permite visualizar simultáneamente el cambio en las formas de onda de cada canal del registro y el video. Puede verse claramente la correlación entre los cambios en las ondas, y el movimiento del paciente.

Se construyeron componentes para la visualización 2D de los registros EEG que permiten aplicar diversos algoritmos de interpolación sobre la magnitud asociada a las señales registradas (no necesariamente el voltaje) que se quiera representar.

Se crearon componentes para mantener la sincronización entre las diferentes ventanas de visualización (1D, 2D, línea de tiempo, ventana de eventos y video) y su funcionamiento es el esperado.

Se diseñó una interfaz de usuario estándar amigable y similar a los programas comerciales que permite que quienes diseñen los algoritmos se enfoquen en la implementación propia del algoritmo, y no a la laboriosa tarea de comunicación con el usuario. A su vez, esta interfaz permite que el especialista no deba aprender a manejar una herramienta nueva, cada vez que se adaptan nuevos algoritmos.

Se crearon componentes que permiten seleccionar los diferentes montajes usuales en el registro EEG, e inclusive se admite la creación de nuevos montajes. Dentro del montaje se puede definir las escalas y los colores para cada una de las señales, los canales que la conforman (para montajes

bipolares) y la aplicación o no de filtros digitales (cuyas frecuencias de corte son seleccionables), en la Figura 25 aparece la interfaz con el usuario para este propósito. También se puede visualizar la ubicación de los electrodos en la representación topográfica, ver Figura 26.

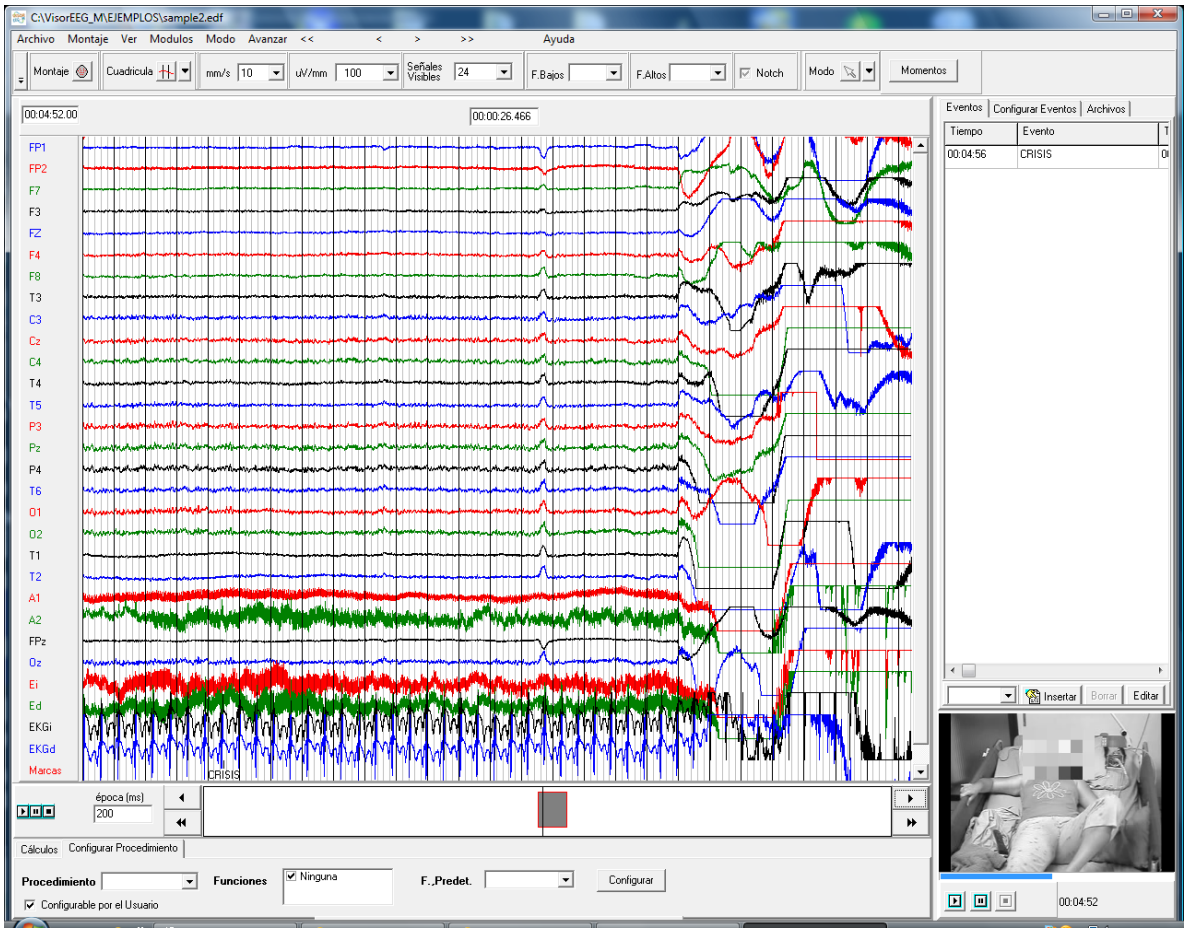


Figura 24. Registro de un episodio epiléptico.

Se incorporó la posibilidad de incluir módulos de procesamiento y análisis elaborados en lenguajes de programación de alto nivel. Estos módulos pueden ser binarios (archivos DLL), o scripts en *Matlab*® e inclusive queda abierta la posibilidad de incluir módulos en otros lenguajes si se implementan apropiadamente los respectivos intérpretes.

Se hicieron las pruebas básicas de la interfaz para comunicación con módulos implementando algunos módulos básicos y las pruebas muestran que el funcionamiento es el esperado, tanto para los módulos binarios como para los módulos escritos en *Matlab*®.

Un resultado que cabe resaltar, a pesar de que no estaba propuesto dentro de los objetivos, fue el registro ante la dirección nacional del derecho de autor del programa *Visor_EEGM.*

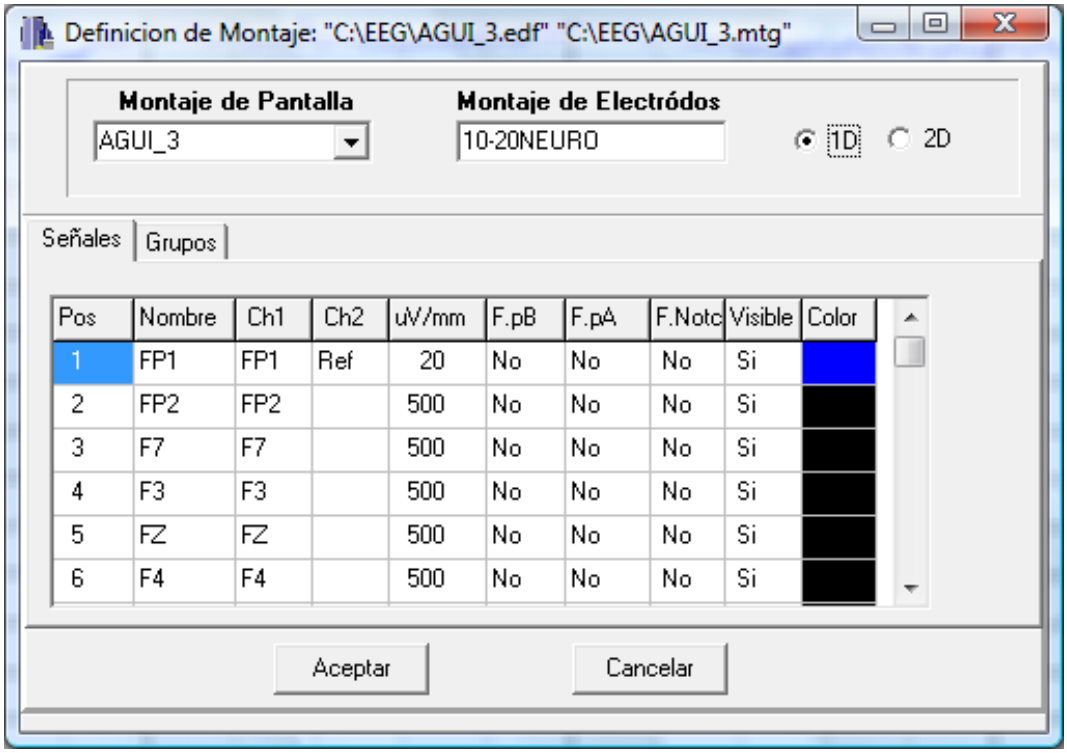


Figura 25. Configuración de las señales que hacen parte de un montaje.

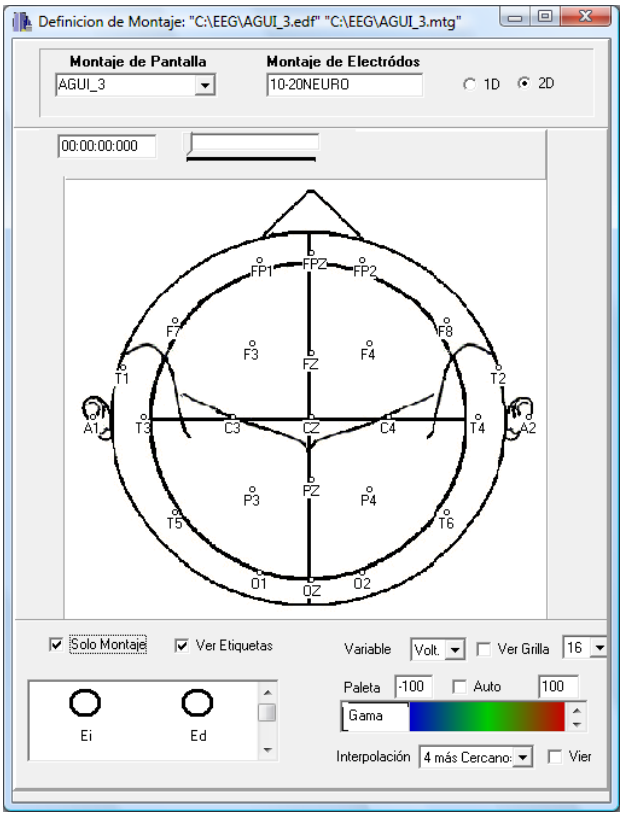


Figura 26. Configuración de la posición de los electrodos en la visualización topográfica

6. CONCLUSIONES Y TRABAJOS FUTUROS

Con el desarrollo de este proyecto se logró crear una interface para acceder a los registros de EEG y presentar las respectivas señales conforme a la forma usual que lo hacen los programas comerciales, con la ventaja de tener a disposición dicha interface para futuras implementaciones. Acorde con este avance, se agregó la posibilidad de utilizar un mismo entorno para que quienes desarrollan los algoritmos de procesamiento de señales utilicen dicha interfaz en sus proyectos, sin la penosa tarea de diseñar, probar y depurar la parte de la comunicación con el usuario.

Para este cometido se creó un sistema de componentes que permiten que el programa invoque rutinas que se pueden ir incorporando al programa sin la necesidad de recompilar todo el programa pues solo se deben codificar las nuevas rutinas y/o controles en un módulo independiente cumpliendo con los prototipos que se propusieron y agregarlos a las trayectorias específicas donde el programa busca dichos módulos a la hora de ejecutarse. Adicionalmente, se complementó con la posibilidad de agregar módulos que estuvieran codificados en algún lenguaje interpretado si previamente se le proporciona al programa (también como otro módulo) un sistema que permita interpretar dicho lenguaje; utilizando este sistema se incorporó un intérprete de Matlab® para tener la posibilidad de ejecutar funciones en scripts de tal lenguaje.

La visualización topográfica, ya sea de las señales, o de variables calculadas a partir de éstas, permite ver de una forma más ilustrativa la distribución en la corteza cerebral para la asociación posición-actividad cerebral. Para el cálculo de los valores en las coordenadas que no son electrodos se está haciendo una interpolación de los N-Cercanos, pero queda abierta la posibilidad de codificar otras funciones utilizando nuevos módulos. Adicionalmente ya se ha empezado a codificar una versión 3D de visualización topográfica por recomendación de los especialistas del Neurocentro.

En el momento las API se han probado con un conjunto reducido de funciones, procedimientos y controles implementados tanto en módulos binarios como en módulos de *Matlab*® y dichas pruebas demuestran que la interface funciona eficientemente. Deben hacerse algunas pruebas con más módulos para determinar su eficiencia en estas condiciones. En el momento se están haciendo los contactos con los investigadores de las universidades de la región y con la colaboración del Instituto Neurocentro de Pereira se seleccionarán algunas rutinas de preprocesamiento de señales: como detección de artificios por movimiento de los ojos; y procesamiento: caracterización y clasificación de señales que ya se han implementado en *Matlab*® en proyectos previos para hacerles las respectivas adaptaciones para invocarlas en módulos en *Matlab*® o la respectiva recodificación en lenguaje C para implementarlas en módulos binarios.

Además de la utilización directa del visor, otros proyectos se beneficiarán de algunas de las partes desarrolladas, por ejemplo, se están haciendo las adaptaciones necesarias a las rutinas para cargar módulos binarios y en *Matlab*® para el proyecto de Maestría en Ingeniería Eléctrica Línea de Instrumentación y Control "Diseño de un sistema de Entrenamiento para la Detección de Zonas Cerebrales en la Cirugía de la Enfermedad de Parkinson" del estudiante José Bestier Padilla.

BIBLIOGRAFÍA

[1] Mauricio Orozco A. Julio Fdo. Suarez C. Álvaro Ángel Orozco, 2009. Entrenamiento de sistemas en la identificación Automática de Patologías. Cesar Germán Castellanos. Centro de Publicaciones UTP.

[2] Leif Sornomo and P. Laguna, 2005. Bioelectrical Signal Processing in Cardiac and Neurological Applications, Elsevier Academic Press.

[3] Lopes da Silva and Ab. van Rotterdam, 2005. Biophysical aspects of EEG and Magnetoencephalogram generation, in Electroencephalography. Basic principles, clinical applications and related fields, Editores :Niedermeyer and F. Lopes da Silva.

[4] Ernst Niedermeyer. Discovery of Electrical Phenomena. Electroencephalography , 5th Edition. Editors: Niedermeyer, Ernst; da Silva, Fernando Lopes.

[5] Quiter PM, McGillivray BB,Wadbrook DG, 1977. The removal of eye movement artifact from the EEG signals using correlation techniques. Random signal analysis. IEEE Conference Publication; 159: 93-100

[6] Verleger R, Gasser T, Mocks J, 1982. Correction of EOG artifacts in eventrelated potentials of the EEG: aspects of reliability and validity. Psychophysiology 19: 472-80.

[7] Berg P, Scherg M, 1991. Dipole models of eye movements and blinks. Electroencephalography Clinical Neurophysiology 79. 36-44.

[8] Barkley GL, Baumgartner C. MEG and EEG in epilepsy, 2003. Journal of Clinical Neurophysiology 20. 163-78.

[9] Hinterberger T,Weiskopf N,Veit R,Wilhelm B, Betta E, Birbaumer N, 2004. An EEG-driven brain-computer interface combined with functional magnetic resonance imaging (fMRI). IEEE Trans Biomed Eng 51. 71-74.

[10] http://www.neurosky.com. [Consulta: Octubre 2010]

[11] http://www.emotiv.com. [Consulta: Octubre 2010]

[12] O'Leary, J. L., and Goldring, S., Science and Epilepsy, Raven Press, New York, 1976, pp. 19–152.

[13] Saeid Sanei and J.A. Chambers. 2007. EEG SIGNAL PROCESSING. Centre of Digital Signal Processing. Cardiff University, UK.

[14] Ramiro Arango., José Bestier Padilla B, 2009. Artículo: VISOR EEG_M: HERRAMIENTA COMPUTACIONAL PARA LA VISUALIZACIÓN Y EL ANÁLISIS DE ELECTROENCEFALOGRAMAS. Revista de investigaciones de Uniquindío.

[15] Erwin-Josef Speckmann Christian E. Elger. 2.009. Introduction to the Neurophysiological Basis of the EEG and DC Potentials. 2da Ed. Urban & Schwarzenberg,Baltimore.

[16] M. Bear, B. Connors, and M. Paradiso, 1996. Neuroscience: Exploring the Brain, Williams and Wilkins,Baltimore.

[17] Ernst Niedermeyer. 2004. Sleep and EEG. Electroencephalography , 5th Edition. 194-207.

[18] Milos Matousek. 1991. EEG patterns in various subgroups of endogenous depression. International Journal of Psychophysiology. Volume 10.

[19] E.˜Niedermayer, "Abnormal EEG patterns: Epileptic and paroxysmals," in Electroencephalography.

[20] Blume WT, Dreyfus-Brisac, 1982. Positive Rolandic Sharp Waves in Neonatal EEG.

[21] Clancy y Tharp, 1996. Positive temporal sharp waves in electroencephalograms of the premature newborn. Neurophysiologie Clinique/Clinical Neurophysiology Volume 26, Issue 6.

[22] Haas, L F, 2003. "Hans Berger (1873-1941), Richard Caton (1842-1926), and electroencephalography."

[23] Eckart O. Altenmaller Thomas F. 2004. Mante Christian Gerloff. Neurocognitive Functions and the EEG. Electroencephalography , 5th Edition. 661-683.

[24] Genaro Daza-Santacoloma, Julio Fernando Suárez-Cifuentes y Germán Castellanos-Domínguez, 2009. Preproceso de datos en bioseñales: una aplicación en detección de patologías de voz, Revista Ingeniería e Investigación. Ing. Investig. v.29 n.3. 92-96.

[25] Saeid Sanei and J.A. Chambers, 2007. EEG Signal Processing. Centre of Digital Signal Processing Cardiff University, UKEEG SIGNAL PROCESSING, Copyright John Wiley & Sons Ltd.

[26] Itil, K.Z. 1993. Responses to the views and commentary on Standard Specification for Transferring Digital Neurophysiological Data Between Independent Computer Systems. J. Clin. Neurophysiol. 10:535-536.

[27] Gregory L. Krauss et All. Digital EEG.Gregory L. Krauss W. Robert S. Webber. Electroencephalography , 5th Edition. 797-815.

[28] Carlos Platero. Introducción al Procesamiento Digital de Señales. Electrónica Industrial. Universidad Politecnica de Madrid. Disponible en: www.elai.upm.es/spain/Publicaciones/pub01/intro_procsdig.pdf

[29] Juan Vignolo Barchiesi Procesamiento Digital de Señales. Ediciones Universitarias de Valparaíso. Pontificia Universidad Católica de Valparaíso.

[30] Takashi Yahagi. An Overview of the History and Development in Digital Signal Processing. Second International Conference on Electrical and Computer Engineering ICECE 2002, 26-28 December 2002.

[31] www.ni.com National Instruments. [Consulta: Octubre 2010].

[32] Niedermeyer, E. 2003b. The clinical relevance of EEG interpretation Clinical Electro-encephalography. 34-38.

[33] R. Schneider, A. Bauer, 2004. libRASCH: A Programming Framework for Signal Handling, IEEE Computers in Cardiology**.** 53-56.

[34] Arnaud Delorme, Scott Makeig, 2004. EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. Swartz Center for Computational Neuroscience, Institute for Neural Computation, University of California San Diego, La Jolla, CA 92093-0961, USA. Journal of Neuroscience Methods. 134.

[35] ROMERO, Abel, JUGO, Diego y PARADA, Marco, 2007. Diseño e implementación de un instrumento virtual para la adquisición y procesamiento de señales fisiológicas. INHRR, vol.38, no.1, ISSN 0798-0477. 11-19.

[36] Pablo A. Cardona, Vladimir Mayoral, Pablo A. Muñoz, 2006. Grupo GAMA7—gama@uniquindio.edu.co CEIFI. Sistema para el Registro y Visualización de Doce (12) Derivaciones de un Electrocardiograma (ECG), Universidad del Quindío.

[37] Sayra M. Cristancho S. Carlos D. Giraldo T. Alex A. Monclou S, 2008. Integración de la Adquisición y Visualización de Señales Biomédicas: BIOLAB. Universidad Pontificia Bolivariana. Colombia. RevistaeSalud.com. Vol 4, No 15.

[38] http://www.natus.com/index.cfm?page=products_1&crid=46&contentid=102. [Consulta: Octubre 2010].

[39] http://www.eeg-persyst.com/index.php/products. [Consulta: Octubre 2010].

[40] Herbert Schildt, 1997. BORLAND C ++. MANUAL DE REFERENCIA Editorial McGraw-Hill. 150-160.

[41] http://www.embarcadero.com/products/cbuilder. [Consulta: Octubre 2010].

[42] George Kamberov. Computer Graphics CS537 A Frame Buffer Primer. Double Buffering and Animation Basics. Stevens Institute of Technology, Hoboken, NJ 07030, USA. Disponible en: http://www.cs.stevens.edu/~kamberov/teach/S2007/537/notes/buffers537.pdf

[43] http://www.videolan.org. [Consulta: Octubre 2010].

[44] Harold Howe. 2004. BCB Developers: Articles. Using Visual C++ DLLs in a C++Builder Project. 2-7.

[45] MOORE, HOLLY, 2009. MATLAB PARA INGENIEROS. Editorial Pearson. ISBN: 9702610826. 10-12.

[46] Juan Vignolo Barchiesi, 2008. Introducción al Procesamiento Digital de Señales. Ediciones Universitarias Valparaiso. Pontificia Universidad Católica de Valparaiso. 22-23.

[47] MATLAB® Compiler™ 4 User's Guide. Disponible en: www.mathwork.com. [Consulta: Octubre 2010]

ANEXO 1. PROCESO DE CONSTRUCCIÓN DE UN MODULO PARA EL VISOR EEG_M

DEFINICIÓN

Un módulo consiste en una sección de programa creada y/o compilada de forma independiente del *VisorEEG_M*, pero que puede incorporarse a éste, si cumple con los prototipos que se detallarán a continuación.

Un módulo puede ser visto como una serie de funciones y/o controles que implementan algoritmos para ciertos procesos y que pueden ser invocados desde el *VisorEEG_M*; estos módulos pueden estar codificadas en lenguajes que generen código ejecutable como lenguaje C o en lenguajes interpretados como *Matlab*®. Un módulo escrito en un lenguaje compilado también puede contener referencias a funciones que podrán estar contenidas en el *VisorEEG_M* u otros módulos (a estas referencias se les denomina en éste documento como procedimientos).

Si el módulo está codificado en lenguaje C, C++ o cualquier lenguaje que genere código ejecutable, éste se debe compilar en un archivo con extensión .DLL y deberá ubicarse en la carpeta *modulos* de la aplicación; estos módulos también pueden definir procedimientos que pueden invocar funciones de otros módulos. Un modulo puede acceder a los datos de un registro EEG por medio de una serie de funciones que están implementadas en el *VisorEEG_M*.

Si un módulo tiene las funciones codificadas en un lenguaje interpretado como *Matlab*®, éstas se deben ubicar en una subcarpeta de la carpeta *modulos\lenguaje*, cuyo nombre será el nombre del modulo. Para la ejecución de estas funciones, el PC donde se ejecuta el *VisorEEG_M* deberá tener instalado todo el soporte para la interpretación de las funciones (en el caso de *Matlab*®, el RTL o ***Runtime Library***). La forma como los módulos de cada tipo de lenguaje interpretado se incorporan al *VisorEEG_M* deberá definirse en un archivo con extensión .DLL ubicada en la misma carpeta *modulos\lenguaje* con el nombre *soporte.dll*, implementando una serie de funciones que se describen más adelante.

MODULOS ESCRITOS EN LENGUAJE C

Si se desea implementar completamente el módulo partiendo desde cero se deben implementar las estructuras y funciones definidas en el archivo dlldefs.h. Cuando se cree el archivo DLL se debe exportar la función *GetLibrary_Desc* con la sintaxis de lenguaje C; si el módulo contiene sentencias en C++ se deben usar las palabras reservadas *extern "C" __declspec(dllexport)* en la declaración (en el archivo dlldefs se declara en el macro APIFUNCTION). Además si se utiliza el compilador CBuilder5 se debe activar la opción *treat enum types as int*, dentro de las opciones de compilación (se prefirió esta opción para que hubiera mayor compatibilidad con los otros compiladores comerciales). Para simplificar el proceso de construcción de módulos a quienes utilicen el compilador *C++Builder*, se ha escrito un proyecto ejemplo denominado *SampleModDLL*, que se propone como plantilla para crear cualquier módulo DLL.

A continuación se describen los archivos que hacen parte del proyecto y luego se detalla el proceso de construcción de módulos utilizando la plantilla. Se recomienda que, cada que se desee hacer un

módulo, se copie la carpeta SampleModDll en una nueva carpeta y se hagan las modificaciones del caso.

El proyecto ejemplo consiste en: un archivo .bpr de CBuilder5, que está configurado para crear archivos DLL; un archivo UnitDLL.h que contiene la declaración de las variables globales usadas en el módulo; y dos archivos .c: UnitDLL .c y SampleModDll.c. Este último archivo se debe cambiar por el archivo que contenga las nuevas funciones, procedimientos, etc. (Se recomienda que use este archivo como plantilla y después de hacer las modificaciones guardarlo con otro nombre). Además se pueden agregar otros archivos al proyecto de acuerdo a las necesidades de cada módulo; el proyecto solo tiene dos .c por simplicidad. En el archivo UnitDLL.c se definen como mínimo las siguientes dos funciones:

int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void lpReserved);*

plugin_infoDll GetLibrary_Desc(void* Handle, EJECUTAPROCFUNC EjProc);*

Y opcionalmente:

int InitLibrary(void);

La primera de ellas es común a toda librería DLL y la segunda es la que permite al programa principal enterarse de las funciones, procedimientos y controles contenidos en la librería (así mismo con los parámetros que se le envían a la función quien escribió el módulo dispone de dos mecanismos para invocar a los procedimientos, proceso que se explicó en la arquitectura del visor). En la implementación de esta función en la plantilla se utilizan una serie de variables que declaran en *UnitDLL.h* y se definen en *SampleModDLL.c* (o el archivo que lo remplace), para que contengan todas los datos necesarios para definir la estructura *plugIn_infoDLL* que debe retornar la función *Get_Library_Desc* que se implementa en *UnitDLL.c*. Estas variables consisten en una descripción del módulo y una serie de arreglos de punteros a las estructuras de cada tipo que contiene (un arreglo de controles, uno de procedimientos, otro de funciones, etc).

La tercera función es opcional y será invocada al final del proceso de inicialización de la librería, una vez el programa principal conozca los punteros de objetos contenidos en la librería. Para que se puedan compilar cualquier tipo de módulos se usan las siguientes constantes que definen que tipo de estructuras contiene el módulo (estas constantes se definen en *SampleDLL.h*):

HAY_FUNC	Indica que el módulo contiene funciones
HAY_PROC	Indica que el módulo contiene procedimientos
HAY_CONTROLES	Indica que el módulo contiene controles
HAY_TIPOS	Indica que el módulo contiene descripciones de tipos de datos
HAY_CONVERSORES	Indica que el módulo contiene conversores a lenguajes interpretados

La idea es poner la línea:

#define HAY_FUNC

Sólo, si en el módulo que se está construyendo, hay por lo menos una función; de lo contrario se debería comentar la línea o eliminarla, así mismo deberá escribirse en el módulo .c la respectiva lista de punteros (*funcion_def *Funciones[]=…;*). De manera similar para las otras constantes. Esta declaración debe hacerse en el archivo *SampleDLL.h* o su equivalente si desea ponerle otro nombre, de ahora en adelante se llamarán modulo.h al archivo de definiciones y *modulo.c* al archivo donde se definen las variables.

Los arreglos de punteros para cada tipo de estructura son los siguientes

```
#define HAY_FUNC            //en archivo modulo.h
funcion_def *Funciones[]=…; //arreglo con punteros a las funciones definidas en el módulo

#define HAY_PROC //en archivo modulo.h
Procedimiento *Proc[]=..;//arreglo con punteros a cada uno de los procedimientos del módulo.

#define HAY_TIPOS//en archivo modulo.h
Tipos_Datos *Tipos[]=…;//arreglo con punteros a cada una de las descripciones de tipo.

#define HAY_CONTROLES        //en archivo modulo.h
Control_Def  *Controles[]=…;  //arreglo con punteros a cada uno de los controles.

#define HAY_CONVERSORES        //en archivo modulo.h
Control_EEG  *Controles[]=…;       //arreglo con punteros a cada uno de los controles.
```

Además de definir apropiadamente las constantes y los arreglos de punteros, se deben a definir las variables que identifican el módulo, de la siguiente manera:

```
char    *Version=…;              //versión de la libreria
char    *Autor=…;                //Autor de la libreria
char    *Descripcion=…;          //Descripcion
TipoLib tipoLib=…;                     //tipo de librería
```

Nota: Los nombres de las variables deben ser exactamente como se denominan aquí; de lo contrario, aparecerán errores de variables sin definir cuando se compile el archivo UnitDLL.c.

A continuación se describen cada una de las estructuras y arreglos que pueden hacer parte de un módulo DLL y más adelante la forma de declararlos utilizando el proyecto ejemplo.

FUNCION_DEF

Una definición de función consta de un puntero a una función, junto con unos campos que permiten describir dicha función para el programa y/o para el usuario. La definición de la función se hace utilizando las siguientes dos estructuras:

```
struct Descripcion_F    //descripcion de una funcion o un procedimiento al cual se le asigna una
//funcion
{
char tipoInputStr[50];          //cadena con nombre del tipo de variable en los parametros
                                //(si son varios debera ser una estructura)
char tipoOutputStr[50];         //cadena con nombre del tipo de variable en retorno
                                //(si son varios debera ser una estructura)
char Source_Name[240];          //cadena con nombre del archivo en el que esta escrito
                                //(normalmente se le asocia __file__)
 char UserName[30];             //nombre con el que el usuario vera a la funcion
 char Name[27];                 //nombre cn el que los modulos veran a la funcion
 char Descripcion[100];         //breve descripcion de la funcion (para el usuario)
 TipoFunC  tipoFunc;            //tipo de funcion: dll_func (para las funciones escritas en dll)
};
struct funcion_def              //estructura que contiene descripcion y puntero (o indice en
                                //caso de ser interpretada) donde esta definida la funcion

{
  Descripcion_F D;
 union
 {
  int (*PFunc)(void* p,void* r);  // puntero a la funcion retorna :0 si se ejecuto //correctamente,
                                  //otro valor si hubo error
  int m_index;                    // en Matlab®: indice de nombre de la definicion Matlab® en la
                                  //clase contenedora
 };
 void *Propietario;               // reservado para el uso interno (puntero a objeto modulo que lo
                                  //contiene)
```

PROCEDIMIENTO

Un procedimiento consiste en una estructura que contiene un índice que determina la función a utilizar (índice de un arreglo tipo *FUNCION_DEF*), la descripción del procedimiento, junto a una descripción del tipo de dato de sus parámetros, retorno y opciones. En algunos procedimientos la estructura puede contener un puntero a la función que despliega el formulario utilizado para establecer los parámetros y/o un puntero a la función que despliega el formulario para mostrar los resultados de evaluar dicho procedimiento.

Existen dos tipos de procedimientos: los procedimientos definidos en el visor y los procedimientos definidos en módulos. La cantidad de procedimientos definidos en el visor sólo podrá ser modificada cuando se compila el visor, pero la cantidad de procedimientos definidos en los módulos podrá ser ampliada por cualquier módulo.

Cada procedimiento podrá configurarse para que invoque alguna de las funciones contenidas en los módulos que coincida en sus parámetros, valores de retorno y opciones contenidas en los módulos construidos para tal fin. Cuando se define la función se establece si el usuario podrá elegir dicha

función en el momento de ejecución o solo podrá hacerse a través de los controles contenidos en los módulos (o inclusive siempre apunta a la función por defecto, si no se agregan controles para elegir la función).

El tipo de dato para definir un procedimiento se llama ***Procedimiento*** y su declaración es:

```
struct Procedimiento                //estructura que contiene la definicion de un procedimiento
                                    //al que se le puede asociar una funcion
{
  Descripcion_F  D;          //descripcion del procedimiento
  bool        UserCnf;       //1 si el usuario puede configurarlo
  bool        (*ConfiguraProc)(Procedimiento* P);
                              //funcion que configura el procedimiento
                             //en caso de ser configurable por usuario)
  void        (*ShowResult)(Procedimiento* P);     //funcion que muestra resultado de evaluar
                             //el procedimiento (en caso de ser configurable por usuario)
  void         *Propietario;     //propietario (para uso interno)
  int         FuncionIndex;    //indice de la funcion a la que apunta
  int        *PosiblesFIndex;  //funciones que coinciden con el formato del procedimiento
                               //(para desplegar opciones)
  int         nPosiblesF;      //cantidad de posibles funciones

};
```

ASOCIACIÓN PROCEDIMIENTO FUNCION

Cuando se quiere tener un procedimiento que inicie apuntado a una función por defecto ya sea que el procedimiento sea visible o no, se debe indicar en el módulo con una estructura tipo *Proc_Funcion_Asociacion*, y este puntero debe aparecer en la lista *ProcFunctions*. Esta estructura debe contener un puntero al procedimiento, seguido por dos punteros a carácter que identifican a la función a la que se quiere apuntar, el primero apunta al nombre del módulo y el segundo con el nombre de la función.

```
struct Proc_Funcion_Asociacion
{
 struct Procedimiento* P;       //PUNTERO al PROCEDIMIENTO
 char  *Func_Name;              //Nombre de la FUNCIÓN
 char  *Module_Name;            //Nombre de MODULO que contiene el PROCEDIMIENTO
};
```

CONTROL_DEF

Por cada control que contenga un módulo se debe definir una estructura de tipo *Control_Def*; en dicha estructura se proporciona el nombre del control, el nombre que indica en que panel de controles se debe ubicar, una breve descripción de la utilidad de dicho control, un puntero a donde están los datos que define el control (habitualmente este puntero será un puntero a un objeto visual,

como por ejemplo un *TButton* de la VCL), un puntero a una función para invocar cuando el control se agregue al respectivo panel y por último un puntero a la función que se debe invocar cuando se quiera destruir el panel, el puntero Propietario es de manejo interno de la clase contenedora.

```
struct Control_Def          //ESTRUCTURA QUE CONTIENE LOS DATOS NECESARIOS
                            //PARA MOSTRAR UN PANEL EN EL VISOR
{
 char Panel[50];            //NOMBRE DEL PANEL
 char Nombre[50];           //NOMBRE DE LA HERRAMIENTA PARA EL USUARIO
 char Descripcion[200];     //descripción del control
void *Datos;                //puntero a donde estan los datos
 int (*Set_Parent)(void* datos,void* Handle,void* WinParent); //funcion se invoca al crear el
                            //control. Debe retornar 0 si el control se pudo cargar u otro
                            //valor en caso contrario
 int (*Destroy)(void *Handle);    //funcion que se debe invocar para destruir el control
void *Propietario;          //propietario (para uso interno)

};
```

CONSTRUCCIÓN DE UN MODULO PARA EL EEGVIEW EN EL BORLANDC BUILDER A PARTIR DEL ESQUEMA BÁSICO

En esta sección de este documento se describen los pasos necesarios para la construcción de un módulo. El video ***construccionModulo*** muestra cada uno de los pasos descritos en esta guía.

1. Cree una carpeta con el nombre del módulo y guarde en ella todos los archivos que piensa hacer parte del proyecto y los que construya durante el proyecto.

2. En el BorlandC-Builder elija la opción del menú, Nuevo y escoja la opción ***DLL-Wizzard***, tal como aparece en la Figura A:

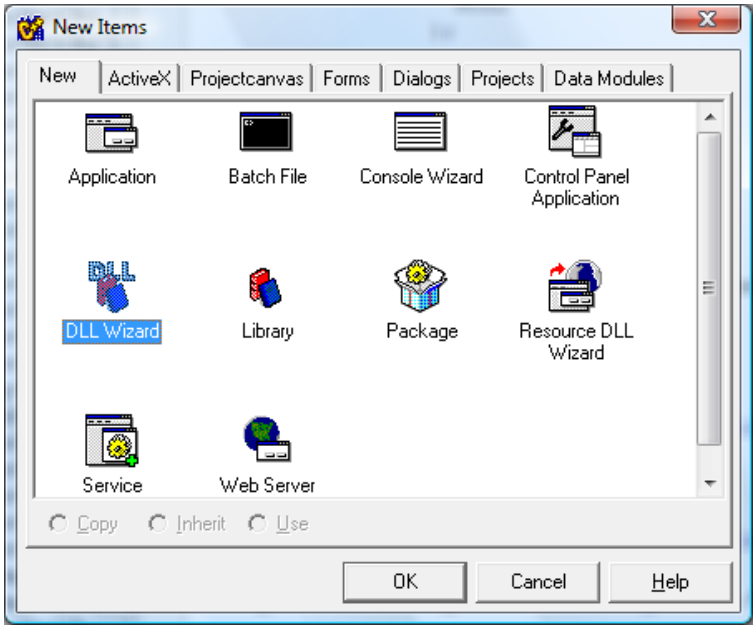


Figura A. Cuadro de dialogo para crear una nueva DLL

2.1 Asegúrese de que el proyecto compile con la opción ***Treat enum types as ints***, ver Figura B.

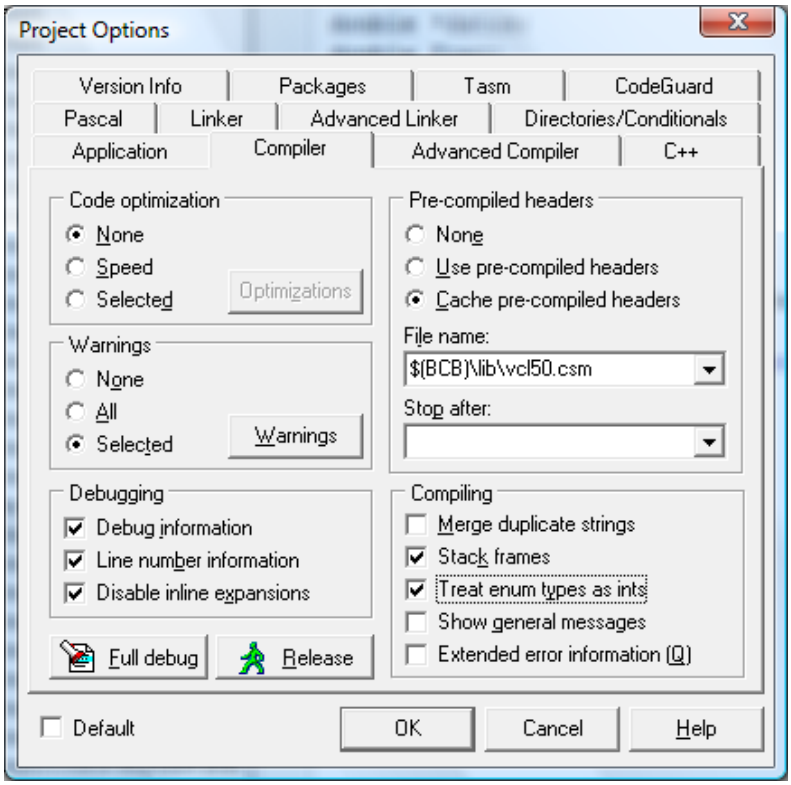


Figura B. Cuadro de diálogo para escoger los ENUM como enteros

3. Elimine el archivo ***Unit1.cpp*** y agregue el archivo ***DllUnit.cpp***. Después de incluirlo grábelo con otro nombre en la carpeta del proyecto.

4. Separe en uno o varios archivos con extensión .cpp y .h las funciones que desea exportar al visor y agréguelos al proyecto. Estas deben ser declaradas preferiblemente utilizando algunos de los tipos básicos declarados en ***modulos_struct.h***. Si su módulo define algún procedimiento agregue al proyecto también el archivo que lo define. Agregue la línea:

   #include "nombreModulo.h"

   Donde nombreModulo es un archivo que contiene las declaraciones de las constantes que indican lo que contiene el archivo (HAY_...).

   Escriba el archivo modulo.h. Por ejemplo:

   ```
   #ifndef _NOMBRE_MODULO_
   #define _NOMBRE_MODULO_

   #define HAY_FUNCIONES       //si en el modulo hay declaradas funciones para exportar
   #define HAY_PROCEDIMIENTOS  //Si en el modulo hay declaración de procedimiento
   #define HAY_TIPOS           //Si en el archivo hay declaración de tipos

   #endif
   ```

   Los pasos 5 a 9 son necesarios si el módulo contiene funciones exportables. Los pasos 10 y 11 son necesarios si el módulo contiene procedimientos. Los pasos 12 y 13 son necesarios si el programa define una nueva representación de tipo de dato. Y los pasos 14 y 15 son necesarios si el programa define por lo menos un control.

5. Verifique que los parámetros de las funciones a exportar se pueden representar con una de las estructuras exportables definidas en ***modulos_struct.h***. Igualmente, verifique que el tipo de dato retornado por la función se puede representar con un tipo de dato o estructura del mismo archivo. Si no existe una estructura que represente los datos apropiadamente (ya sea de los parámetros y/o tipo de retorno), se debe crear y registrar dicha estructura (proceso que se explicará más adelante en este documento), y se asume que hay uno o varios procedimientos que tienen las mismas estructuras en los parámetros y en el valor de retorno que debe estar declarado en el modulo actual o en otro módulo (de otra forma la función nunca será invocada desde el visor).

6. Por cada función a invocar desde el visor declare una función de interfaz con el prototipo:

   int ***NombreFuncion****(void *p,void* r);*

   Donde :

p Puntero a los parámetros de la función.
r Puntero al valor de retorno de la función.

Esta función será la que realmente se invoque desde el VISOR cuando se utilice dentro de un procedimiento, así que implícitamente en ella se debe llamar a la función que se escribió originalmente.

7. Dentro de la definición de la función utilice un puntero a las estructuras usadas o creadas para acceder a cada uno de los parámetros, valores de retorno y/o las opciones. Así:

```
int NombreFuncion(void *p,void* r)
{
 tipoParametro *Param;
 tipoRetorno     *Retorno;
 tipoOpciones  *Opcion;
     //acceso al primer parametro   Param->Campo1....
 ....
   //invocar a la funcion
  Funcion(Param->Campo1....)
}
```

Ejemplo:

```
double get_min(double *d, int n)
{
 ....
}
int Minimo(void *p,void* r)
{
 Datos_Rango* D=(Datos_Rango*)p;
 *(double*)r=get_min(D->datos,D->ndatos);
....
```

8. Por cada función a exportar declare una variable de tipo ***función_def*** (declarada en interfaz.h), que contenga la descripción de la función. Ejemplo de descripción de función:

```
funcion_def  F_Minimo={
                {
                "Datos_Rango",     //nombre de la estructura que representa los parametros
                "double",          //nombre del tipo de dato que representa el valor de retorn
                "void",            //n. del tipo de dato que representa opciones de la funcion
                __FILE__,          //Macro de BUILDER con el nombre del archive actual
                "Minimo",          //Nombre de la función para el usuario
                "minimo",          //Nombre de la función para el programa
```

```
"Halla el minimo de un conjunto de datos"//descripción de la función para el
                    // usuario
},
Minimo,             //puntero a la funcion   a invocar
};
```

9. Declare un arreglo de tipo ***funcion_def*** llamado Funciones, que contenga punteros a cada una de las funciones exportables del módulo. Ejemplo:

```
funcion_def *Funciones[]=
          {
            &F_Minimo,
            &F_Maximo,
            &F_Media,
            &F_Varianza,
            &F_DesviacionEstandar,
          };
```

10. Si el módulo contiene Procedimientos defina una variable de tipo ***Procedimiento*** por cada uno de los procedimientos que contenga el módulo. Ejemplo:

```
Procedimiento   P_Ver=
        {"Datos_Rango","CanalyNDatos",__FILE__,
            "Medida en Control1","MedidaControl",
            "Medida que aparece cuando se pulsa control1"
           },
           false, //miembro que indica que el usuario puede modificarlo
           NULL,
           NULL
          };
```

11. Declare un arreglo de tipo ***Procedimiento*** llamado *Procs*, que contenga punteros a cada una de los procedimientos del módulo. Ejemplo:

```
Procedimiento *Procs[] ={ &P_Ver,...};
```

12. Si el módulo contiene Representación de tipos de datos defina una variable de tipo ***Representacion*** por cada uno de los procedimientos que contenga el módulo. Ejemplo:

```
Representacion Datos_Rango_Info={"Datos_Rango","int double* double*
"};//Datos_Rango;
```

13. Declare un arreglo de tipo Procedimiento llamado Tipos, que contenga punteros a cada una de los tipos de datos del módulo. Ejemplo:

```
Info_Struct  Tipos[]={&D_Rango_Info,...};
```

14. Si el módulo contiene controles declare una variable tipo *Control_Def* por cada control. Ejemplo:

    Control_Def Control_1={"ToolBar","Momentos","Calcula los momentos de una serie de datos",
    new TFrame1(NULL),Set_Parent_VCL};

    En éste ejemplo se asume que la clase TFrame1 existe y es precisamente una clase derivada de *TFrame* de las clases VCL de Builder que contiene el control que se quiere hacer aparecer en el Toolbar del Visor. Este frame debe poseer una altura menor o igual a 45 pixeles (si tuviese más altura sería recortada a 45). *Set_Parent_VCL* es la dirección de la función que se encarga de dibujar por primera vez el control en el momento de ponerlo en pantalla. Ésta función está declarada en el archivo *vclcontrols.h* y definida en la librería *vclcontrols.lib* (por lo cual hay que agregar este archivo al proyecto). Si la variable se debe crear dinámicamente se puede hacer dentro de la función *CrearControles* que es invocada por la función *GetLibraryDesc* del archivo *UnitDllModulo*.

15. Declare un arreglo que contenga todos las direcciones de cada uno de los controles :

    *Control_Def *Controles[1]={&Control_1,…};*

16. Defina las variables que identifican al módulo. Ejemplo:

    ```
    char    *Version      ="1.0.0";
    char    *Autor        ="Ramiro Arango";
    char    *Descripcion  ="Funciones Estadísticas";
    TipoLib tipoLib       =TIPOLIB_PROCESS;
    int     nFunciones    = sizeof(Funciones)/sizeof(Funciones[0]);
    ```

17. Una vez compilada sin errores, el archivo con extensión DLL debe ser copiado en la subcarpeta Modulos del subdirectorio donde se instaló el VisorEEG_M. Y si todo estuvo bien las funciones, los procedimientos, controles y los tipos de datos pueden ser vistos y utilizados en la configuración del programa y/o en la interfaz con el usuario. Para verificar que un módulo está disponible en el visor ejecute el programa y elija la opción Modulos->Librerias. El módulo debería aparecer dentro de las librerías disponibles y se deberían ver las funciones, procedimientos y/o controles propios de dicho modulo, tal como aparece en la Figura C.

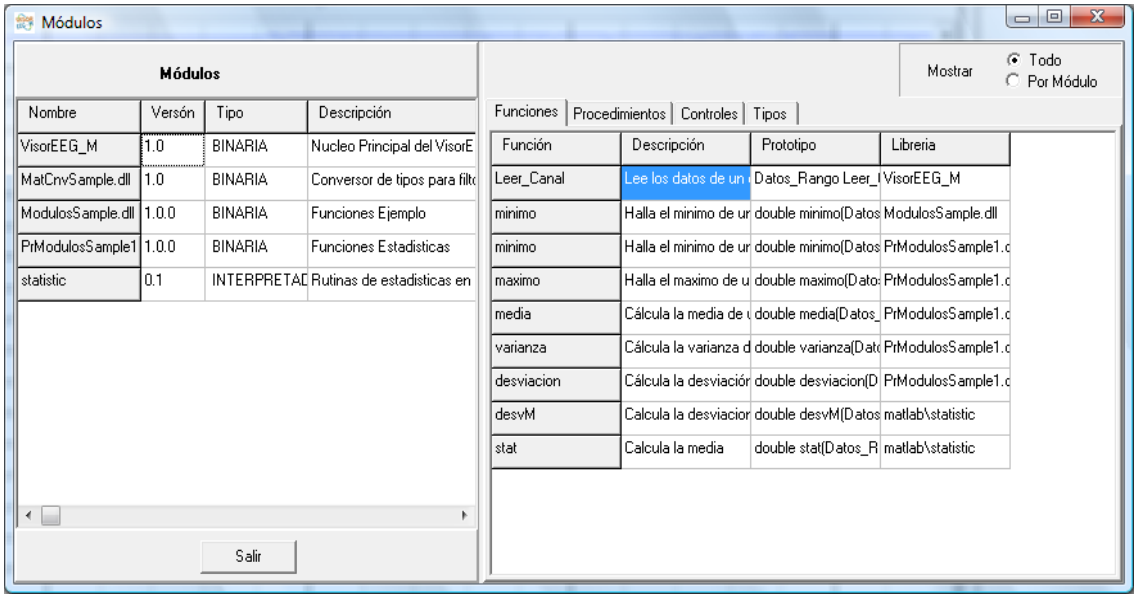


Figura C. Cuadro de dialogo que muestra el contenido de los módulos instalados

MODULOS DE FUNCIONES EN MATLAB®

Un módulo de funciones en *Matlab*® se define como una subcarpeta de la carpeta *modulos\Matlab*® que contiene una serie de archivos .m, en cada uno de los cuales implementa una función que podrá ser invocada desde un procedimiento del programa *VisorEEG_M* o de cualquiera de sus módulos DLL. En esta carpeta también se deberá ubicar un archivo *descripcion.txt* que contiene en cada una de sus líneas la siguiente información:

Línea	Información
1	Autor: *Nombre del Autor del módulo*
2	Descripción: *Texto descriptivo de lo que hacen las funciones*
3	Versión: *Versión del modulo*
4	Fecha: *Fecha de construcción del modulo*

Cada una de las funciones de *Matlab*® deberá ser escrita siguiendo un formato de encabezado para que pueda ser incorporado al *VisorEEG_M*. El encabezado se incorpora al inicio del archivo (a excepción de la primera línea, cada una de sus partes está definida en un comentario *Matlab*® para no interferir en su ejecución por parte del intérprete de *Matlab*®) y deberá contener cada una de las siguientes partes:

function [Variable(s) de retorno] = Nombre de la funcion(paramatro1. Parametro2,…)

% Nombre: Nombre de la función para el usuario final

% Descripcion:Texto descriptivo de la función

% Parametros:_Tipo de dato de lenguaje C o tipo de dato registrado en el EEGView que contiene

todos los parámetros, en caso de ser una función con 1 solo parámetro, este tipo podrá ser un tipo básico de lemguaje C, o de lo contrario deberá ser una estructura que ha sido registrada en alguna parte del programa o de los módulos DLL incorporados.

% Retorno: Tipo de dato de lenguaje C o tipo de dato registrado en EEGVieo que contiene todos los valores de retorno.

Un ejemplo de un archivo .m que hace parte del módulo ***statistic*** aparece a continuación:

```
function [mean] = stat(x,n)
% Media
% Calcula la media y desviacion estandar
% Parametros:Datos_Rango
% Retorno:double
n = length(x);
display(x);
mean = sum(x)/n;
```

El tipo de retorno *double*, indica que para convertir una variable de C a una variable *mxArray* se tiene el conversor respectivo en el intérprete proporcionado en el módulo *Matlab®/soporte.dll,* de hecho, también proporciona los conversores para los tipos: *char*, *int* y *Datos_Rango*. Para los parámetros de la función, se asume que en el mismo orden en que se declaran en la lista de parámetros, primero x y luego n, coinciden con los miembros del tipo de dato *Datos_Rango* que, como ya se mencionó, su conversor está definido en el intérprete.

Si la función que se escribe en *Matlab®* tiene una lista de parámetros o de valores de retorno de la cual no se tenga un conversor de tipo compilado, debe hacerse un módulo binario que proporcione tal conversor, a continuación se describe el procedimiento para tal fin.

MODULOS PARA INDICAR CONVERSORES DE TIPO EN MATLAB®

Un conversor de *Matlab®* al *VisorEEG_M* debe ser declarado utilizando la siguiente estructura:

```
struct mxConvert
{
 char *nameTipo;                    //nombre del tipo en el Visor
```

```
mxArray* (*Create_mxArray)(void *,char *);  //puntero a función que crea variable en Matlab®

void*  (*Create_CVar)(mxArray*,char*);      //puntero a función que crea una variable en C

int (*Set_mxArray)(mxArray*,void *,char *); //puntero a función que copia de C a Matlab®

int (*Set_CVar)(mxArray*,void *,char *);    //puntero a función que copia de Matlab® a C

int (*Free_mxArray)(mxArray*,char *);       //puntero a función que libera memoria de Matlab®

int (*Free_CVar)(void*,char *);             // puntero a función que libera memoria de C

};
```

Las funciones para hacer esta conversión deben utilizar las API de *Matlab*®, para lo cual se debe agregar al proyecto el archivo ***engmodVC3.dll*** que sirve de enlace entre las API de *Matlab*® y módulos binarios con funciones en lenguaje C. La carpeta *MatConvert*, contiene el proyecto ***MatCnvSample.DLL*** que es un ejemplo de módulo binario que sólo contiene los componentes necesarios para hacer un conversor para el Visor. El crear un conversor para un lenguaje en particular sólo será necesario en caso de tener un procedimiento al que se le quiera poner a apuntar a un módulo interpretado (en este caso de *Matlab*®), de lo contrario no es obligatorio hacerlo.

ANEXO 2. DESCRIPCION DE LAS LIBRERIAS DEL PROGRAMA

MODULO.LIB

Descripción: Contiene las declaraciones de los tipos de datos, las estructuras y objetos que permiten hacer la interface entre el programa principal del *VisorEEG_M* y los módulos que se incorporan con archivos DLL (en caso de estar escritos en lenguaje C) o con archivos .m (en caso de estar escritos en *Matlab*®).

Archivos:

Codigo Fuente: Class_Engine.cpp, Debug_unit.cpp, DllDriver.cpp, Utipodato.cpp

Estructura/Clase	Clase base	DECLARACION DEFINICION	DESCRIPCION
struct Descripcion_F		*DLLDEFS.H*	Estructura que contiene cadenas que describen una estructura ***funcion_def***
struct funcion_def		*DLLDEFS.H*	Estructura que contiene el puntero a una función que se define en un archivo DLL o en un archivo .m y que puede ser asignada a un procedimiento.
struct Procedimiento		*DLLDEFS.H*	Estructura para indicar que en una parte del programa se pueden asignar funciones creadas en otros módulos DLLs.
struct plugin_infoDll		*DLLDEFS.H*	Contiene todas las funciones, procedimientos y tipos de datos definidos en un módulo DLL, así como las cadenas que los describen para los usuarios y/o los otros módulos.
LIBRARYDESCFUN		*Module.h*	Tipo de función que debe implementarse en un archivo .DLL para que al ser cargada se comunique con la clase ***Module_Handle*** indicándole si hay funciones, procedimientos y/o tipos de datos.
LIBRARYINIT		*Module.h*	Tipo de función que debe implementarse en un archivo .DLL para que al ser cargada se comunique con la clase ***Module_Handle*** asignando apropiadamente los punteros a los procedimientos, funciones y o tipos de datos.
LIBRARYFUNC		*Module.h*	Tipo de puntero a función que contiene los parámetros, y los valores

			de retorno en puntero y que sirve para implementar cualquier función en C que esté definida en una DLL como la función de una variable ***función_def***
class Module_Handle		*Module.h* *Module.cpp*	Clase encargada de contener las funciones y procedimientos de un modulo.
class Module_Class		*Module.hj* *Module.cpp*	Clase base para cargar un modulo.
class Module_PPal	*Module_Handle*	*DllDriver.h* *DllDriver.cpp*	Clase para definir los procedimientos de un programa.
class Module_Handle_DLL	*Module_Handle*	*DllDriver.h* *DllDriver.cpp*	Clase encargada de controlar las funciones cargadas en un modulo DLL
class Modules_Container		*DllDriver.h* *DllDriver.cpp*	Clase encargada de contener todos los módulos DLL y/o módulos *Matlab*® de un programa
class Module_ClassDLL	*Module_Class*	*DllDriver.h* *DllDriver.cpp*	Clase para cargar un modulo DLL.
class Matlab®_Engine		*Class_Engine.h* *Class_Engine.cpp*	Clase para comunicarse con el ***Matlab® Engine.***
struct Info_Struct		*Utipodato.h* *Utipodato.cpp*	Clase que contiene los nombres y tipos de los campos de una estructura.
class TipoDatoStr		*Utipodato.h* *Utipodato.cpp*	Clase base que contiene la representación de una variable y la forma como se muestra en pantalla.
class TDStr_void	*TipoDatoStr*	*Utipodato.h* *Utipodato.cpp*	Clase que contiene la representación de un tipo de dato *void*.
class TDStr_Struct	*TipoDatoStr*	*Utipodato.h* *Utipodato.cpp*	Clase que contiene la representación de una estructura.
class ManTipoDato		*Utipodato.h* *Utipodato.cpp*	Clase que contiene todas las representaciones de datos registradas por el programa o por módulos DLL.
class DebugProcess		*Debugunit.h* *Debugunit.cpp*	Clase para enviar a un archivo información de la ejecución del programa, funciones invocadas, para emitir mensajes de advertencia y/o de errores.
enum TiposCIndex		*engMat.h* *engMat.cpp*	Constantes usadas para indicar el tipo de dato que contiene una variable tipo ***Tipo_ArrayC***
struct Tipo_ArrayC		*engMat.h* *engMat.cpp*	Estructura para contener arreglos N dimensionales en lenguaje C y que pueden ser traducidos a estructuras ***mxArray*** de *Matlab*®
struct mxConvert		*engMat.h* *engMat.cpp*	Conjunto de funciones usadas para crear, liberar, convertir de C a Matlab® y de *Matlab*® a C, las

			variables de un programa.
struct engFunction		*engMat.h* *engMat.cpp*	Estructura que contiene el prototipo de una función de *Matlab*® que puede ser invocada en un programa en lenguaje C, junto a los datos necesarios para compartir los datos de los parámetros y los valores de retorno.

VIDEOC.DLL

Descripción: Contiene la clase que permite la comunicación entre la librería de *VideoLan* (cuyas funciones están declaradas en *libvlc*), y el programa principal.

Archivos:

Código fuente: *unitvlc.cpp*, *videoclass.cpp*.

Librerías: *libvlc.lib*. (de la organización *VideoLan*)

Estructura/Clase	Clase base	DECLARACION DEFINICION	DESCRIPCION
class vlc_video		*Videoclassdll.h* *Videoclassdll.cpp*	Clase que permite la comunicación entre la librería de ***VideoLan*** y el programa principal.

ENGMODVC2.DLL

Descripción: Contiene la clase que permite la comunicación entre el ***Engine*** de *Matlab*® y el programa principal o los módulos DLL que definen procedimientos y/o funciones.

Archivos: *dllmain.cpp, engmat.cpp, EngModVC2.cpp, stdafx.cpp*

Librerías: *libdfblas.lib, libdflapack.lib, libemlrt.lib, libeng.lib, libfixedpoint.lib, libmat.lib, libmex.lib*

libmwblas.lib. libmwlapack.lib, libmwmathutil.lib, libmwservices.lib, libmx.lib, libut.lib, mclcom.lib

mclcommain.lib, mclmcr.lib, mclmcrrt.lib, mclxlmain.lib. Todas estas librerías pertenecen a ***Matlab® Component Runtime*** *(MCR).*

Estructura/Clase	Clase base	DECLARACION DEFINICION	DESCRIPCION
Engine		*Engine.h(de MCR)*	Puntero que permite comunicarse con *Matlab*® por medio de la interfaz ***Component Object Library*** (COM)
mxArray		*Matrix.h (de MCR)*	Tipo de dato que permite compartir datos entre *Matlab*® y lenguaje C.

enum TiposCIndex		*engMat.h* *engMat.cpp*	Constantes usadas para indicar el tipo de dato que contiene una variable tipo ***Tipo_ArrayC***
struct Tipo_ArrayC		*engMat.h* *engMat.cpp*	Estructura para contener arreglos N dimensionales en lenguaje C y que pueden ser traducidos a estructuras ***mxArray*** de *Matlab*®
struct mxConvert		*engMat.h* *engMat.cpp*	Conjunto de funciones usadas para crear, liberar, convertir de C a *Matlab*® y de *Matlab*® a C, las variables de un programa.
struct engFunction		*engMat.h* *engMat.cpp*	Estructura que contiene el prototipo de una función de *Matlab*® que puede ser invocada en un programa en lenguaje C, junto a los datos necesarios para compartir los datos de los parámetros y los valores de retorno.

GRAFICA1D.LIB

Descripción: Contiene las clases que se encargan de la representación 1D de los registros

Estructura/Clase	Clase base	DECLARACION DEFINICION	DESCRIPCION
class TCanvasObj		*CanvasClass.h* *CanvasClass.cpp*	Clase base para objetos que aparecen en el Canvas de visualización 1D.
class TCanvasSeries	*class TCanvasObj*	*CanvasClass.h* *CanvasClass.cpp*	Clase base para representar objetos que contienen varios puntos de una serie de tiempo (curvas, marcas).
class TCanvasSeriesCurve	*class TCanvasSeries*	*CanvasClass.h* *CanvasClass.cpp*	Clase para representar curvas en representación 1D.
class TCanvasSeriesMarks	*class TCanvasSeries*	*CanvasClass.h* *CanvasClass.cpp*	Clase para representar marcas en representación 1D.
class TCanvasSeriesMarksEvents	*class TCanvasSeriesMarks*	*CanvasClass.h* *CanvasClass.cpp*	Clase para representar marcas con líneas en representación 1D.
class TCanvasSeriesMarksText	*class TCanvasSeriesMarks*	*CanvasClass.h* *CanvasClass.cpp*	Clase para representar marcas con texto en representación 1D.
class TCanvasCursor	*class TCanvasSeries*	*CanvasClass.h* *CanvasClass.cpp*	Clase para representar diferentes cursores que aparecen en representación 1D.

class TCanvasBlock	*class TCanvasSeries*	*CanvasClass.h CanvasClass.cpp*	Clase para representar área sombreada cuando se selecciona un bloque.
class TCanvasGrid	*class TCanvasObj*	*CanvasClass.h CanvasClass.cpp*	Clase para representar cuadrícula sobre rep. 1D.
class TCanvasGridX	*class TCanvasGrid*	*CanvasClass.h CanvasClass.cpp*	Clase para representar cuadrícula horizontal sobre rep. 1D.
class TCanvasGridY	*class TCanvasGrid*	*CanvasClass.h CanvasClass.cpp*	Clase para representar cuadrícula vertical sobre rep. 1D.
class TPanelView		*CanvasClass.h CanvasClass.cpp*	Clase para representar un panel en 1D (la representación consta de varios paneles que se refrescan independientemente).
class TGrafica	*class TPanelView*	*CanvasClass.h CanvasClass.cpp*	Clase para representar un panel con las curvas de las señales.
class TTimeView	*class TPanelView*	*CanvasClass.h CanvasClass.cpp*	Clase para representar la posición el tiempo en modo texto en la rep 1D.
class TDeltaTimeView	*class TPanelView*	*CanvasClass.h CanvasClass.cpp*	Clase para representar el tiempo correspondiente a un pantallazo en la rep 1D.
class TScrollXGrafica	*class TPanelView*	*CanvasClass.h CanvasClass.cpp*	Clase para representar la barra del *Scroll* en X de pantallazo en la rep 1D.
class TScrollYGrafica	*class TPanelView*	*CanvasClass.h CanvasClass.cpp*	Clase para representar la barra del Scroll en Y de pantallazo en la rep 1D.
class TittleSignalsGrafica	*class TPanelView*	*CanvasClass.h CanvasClass.cpp*	Clase para representar los títulos de las señales.
class TFrameDraw1D	*class TFrame*	*UnitFrameDraw1D.h UnitFrameDraw1D.cpp*	Clase que contiene todo el *frame* de la representación 1D.

GRAFICA 2D.LIB

Descripción: Contiene las estructuras y clases para la representación 2D de las señales EEG.

Estructura/Clase	Clase base	DECLARACION DEFINICION	DESCRIPCION
class DatoPFile_Image2d		*UnitElePos.h UnitElePos.cpp*	clase para representar 1 punto que hace parte de una imagen 2d

			y que es leído de un archivo.
struct TPoint2D		*UnitElePos.h*	Un punto en 2D.
struct TFoco2D		*UnitElePos.h*	Estructura que contiene un punto que sirve de base para interpolar (corresponde a la posición de un electrodo.
struct PuntoGrilla		*UnitElePos.h*	Estructura que contiene un punto de la grilla y los puntos mas cercanos para hacer interpolación
struct ViewParam		*UnitElePos.h*	Estructura que contiene todos los parámetros de la visualización 2D.
		UnitElePos.h	
class TFrameElectrodos	*class TFrame*	*UnitElePos.h* *UnitElePos.cpp*	Clase que contiene el frame en el que se visualiza la representación 2D.
class TPaleta		*UnitPaleta.h* *UnitPaleta.cpp*	Clase que representa los datos de una paleta para aplicar en la rep 2D.
class TPaletaDiscreta	*class TPaleta*	*UnitPaleta.h* *UnitPaleta.cpp*	Clase que representa los datos de una paleta con un solo color.
class TPaletaGamma	*class TPaleta*	*UnitPaleta.h* *UnitPaleta.cpp*	Clase que representa los datos de una paleta con un función para calcular color RGB.
class TFramePaleta	*class TFrame*	*UnitPaleta.h* *UnitPaleta.cpp*	Clase que contiene el *frame* que aparece par seleccionar paleta.

EVENTOSL.LIB

Descripción: Contiene las estructuras y clases necesarias para manejar las marcas/eventos que se aparecen en la visualización de los registros.

Estructura/Clase	Clase base	DECLARACION DEFINICION	DESCRIPCION
struct Mark_FilterData		*Evento.h*	Estructura que contiene los datos de un tipo de "filtro" para visualización de eventos.
class Mark_Filter_Node		*Evento.h* *Evento.cpp*	Nodo para el manejo de cada nodo árbol de eventos.
class MarkFilter_Manager		*Evento.h* *Evento.cpp*	Clase para contener el árbol de manejo de eventos
class Mark		*Evento.h* *Evento.cpp*	Clase base que contiene los datos de una marca en el

			registro.
class TMetaMark		*Evento.h* *Evento.cpp*	Clase base que contiene los objetos que crean marcas en el programa en el archivo.
class TMetaMark_Manejador		*Evento.h* *Evento.cpp*	Clase contenedora de todos los tipos de *Metamark*.
class MComentario	*class Mark*	*Evento.h* *Evento.cpp*	Clase para marcas que contienen comentarios.
class TMetaComentario	*class TMetaMark*	*Evento.h* *Evento.cpp*	Clase para crear MComentarios.
class MStarted	*class Mark*	*Evento.h* *Evento.cpp*	Clase para contener marca Start (inicio de registro).
class TMetaMStarted	*class TMetaMark*	*Evento.h* *Evento.cpp*	Clase para crear *MSrtarted*
class MPhotic	*class Mark*	*Evento.h* *Evento.cpp*	Clase para eventos *Photic* (estímulos ópticos durante el registro)
class TMetaMPhotic	*class TMetaMark*	*Evento.h* *Evento.cpp*	Clase para crear *MPhotic*
class Manejador_Marcas		*Evento.h* *Evento.cpp*	Clase para manejar marcas, (insertar, borrar, archivar, etc)
class TFrameEvent	*class TFrame*	*UnitFEventos.h* *UnitFEventos.cpp*	Clase que contiene el *frame* de visualizar eventos.
class TEditEvent	*class TForm*	*UnitEditEv.h* *UnitEditEv.cpp*	Clase para dialogo de cambiar evento.
class TFrameCnfArchiv	*class TFrame*	*UnitFEvArch.h* *UnitFEvArch.cpp*	Clase para manejar archivos donde se guardan los eventos.
class TFrameCnfEvent	*class TFrame*	*UnitFEvCnf.h* *UnitFEvCnf.cpp*	Clase visual para manejar árbol donde se configuran eventos
class TNewEvent	*class TForm*	*UnitNewEvent.h* *UnitNewEvent.cpp*	Dialogo para crear un nuevo tipo de evento.

ANEXO 3. PROCESO DE INSTALACIÓN DEL VISOR EEG_M

Contenidos
---
1. Acerca del producto VISOR EEG_M
2. Requerimientos del Sistema
3. Funciones del VISOR EEG_M versión 1.0
4. Instrucciones de instalación
5. Solicitud mayor información

1. Acerca del producto VISOR EEG_M

El CD contiene:

- Versión completa del producto VISOR EEG_M, que consiste en:

* VISOR EEG_M
* Libreria libRASCH (versión GNU)
* Libreria VideoLan (versión GNU)
* WinHelp
* Manual de Instrucción del Producto
* Ejemplos

2. Requerimientos del Sistema

Los requerimientos mínimos del sistema son:

* Pentium II 450 (MHz)
* 64 MB RAM (128 MB recomendado)
* 400 MB de espacio en el disco duro
* Windows XP, VISTA, W7

3. Funciones del VISOR EEG_M versión 1.0

El VISOR EEG_M permite:
3.1 Visualizar EEG en 1D y 2D
3.2 Ver e incluir marcas
3.3 Ver video
3.4 Configurar opciones de visualización y montajes
3.5 Incluir y ejecutar librerías para el análisis y procesamiento de EEG

Nota: Las librerías para el análisis y procesamiento de EEG no están incluidas en este paquete; son elementos adicionales

4. Instalación del Producto

4.1 El programa de instalación deberá iniciarse automáticamente al insertar el CD. Una vez iniciado el proceso de instalación aparece la ventana de la figura A, dando opción de instalar los programas requeridos (si no están previamente instalados), para el correcto funcionamiento del VISOR EEG_M. Si el proceso de instalación no da inicio automáticamente, vaya a "Mi PC", haga clic sobre la Unidad de CD-ROM, y luego doble clic sobre "Install.exe".

Figura A: Instalador de software de soporte y VISOR EEG_M

4.2 Para Instalar cada aplicación dar click sobre el respectivo icono. Como mínimo se debe tener instalado LibRASH antes de proceder a instalar el Visor EEG_M

4.3 INSTALACIÓN DEL VISOR EEG_M:

4.3.1 Dar click sobre el botón 

. Si desea instalar o reinstalar el VISOR EEG-M desde el CD ejecute desde el explorador E:\setup.exe

4.3.2 Para navegar a través de los pasos a seguir, utilice los botones marcados "Siguiente" y "atrás". El proceso podrá abortarse en cualquier momento haciendo click sobre el botón "Terminar".

4.3.3. El progreso de la Instalación puede observarse en la Barra que aparece durante el copiado de los archivos.

4.3.4. Una vez que el Proceso de Instalación haya concluido con éxito, haga click sobre el Botón "Finalizar".

5. Solicitud mayor información

Si tiene algún problema con la instalación, registro u operación del producto, llámenos al teléfono celular 311 767 4009 o contáctenos por correo electrónico a ramy@uniquindio.edu.co. - jbpadilla@uniquindio.edu.co

REQUERIMIENTOS SOFTWARE

A continuación se relacionan los programas o librerías que se requieren para el correcto funcionamiento del VISOR EEG-M:

* Libreria libRASCH
* Libreria VideoLan

El runtime de Matlab® y C++ se deben instalar si se desean incluir procesos de procesamiento o análisis de selañes EEG

Librería LibRASCH

LibRASCH es una librería para acceder a señales /grabaciones hechas en varios formatos, y cuenta con una interfaz común para las señales independiente del formato utilizado. Adicionalmente, LibRASCH posee soporte para procesar y visualizar las señales (por ejemplo, detecta pulsaciones en un ECG o hace los cálculos de parámetros de variabilidad de ritmo cardíaco). Por ahora el enfoque principal de LibRASCH es manipular las señales biológicas (ECG, EEG, la presión sanguínea); sin embargo, la librería permite manipular cualquier clase de señal.

LibrASCH es GNU, y la versión utilizada en este desarrollo es la 0.8.31

Figura 1.1 Interfaz de LibRASCH

La librería está disponible para
- Linux y
- Windows

LibRASCH está desarrollada en C y tiene interfaces para
- Perl
- Pytón
- Octave para Linux

- Matlab®
- Scilab (por el momento sólo para Linux)

Más información puede ser encontrada en el website LibRASCH en www.librasch.org

Librería VideoLan

Los desarrolladores que conforman el proyecto VideoLAN son un equipo de personas encargadas de implementar aplicaciones GNU para multimedia, en particular el visor EEG_M utiliza las DLL desarrolladas en la interfaz LibVLC.

Proyecto VideoLAN

Un poco de Historia

El proyecto comenzó como un proyecto estudiantil en el instituto frances École Centrale Paris, en 1996. Después de una versión completa reescrita en 1998, se convirtió en código abierto, gracias al acuerdo de integrantes del École Centrale Paris, en 2001.

El proyecto comenzó a motivar a los desarrolladores de software fuera del École. Hoy en día es un proyecto mundial con desarrolladores de 20 países.

Desde 2009, el proyecto es completamente separado de École Centrale Paris, y es respaldado por una organización autónoma sin ánimo de lucro.

En particular, el reproductor de medios VideoLan (VLC) es un reproductor, codificador y emisor de medios libre que permite leer archivos, CDs, DVDs, emisiones de red, tarjetas capturadoras y mucho más.

VLC usa códecs internos y funciona esencialmente sobre toda plataforma popular.

La versón de VLC compliada es la 1.0.1 y usa la interfaz Qt4 y es una licencia GNU

Para mayor información visite la página http://www.videolan.org

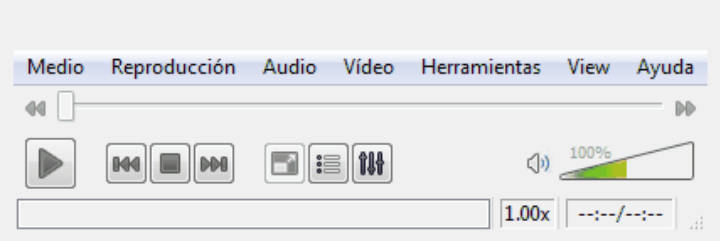


Figura 1.2 Interfaz de VideoLan VLC

WinHelp: Sistema de ayudas de Windows.

Es requerido para sistemas operativos de Windows Vista o Superior.

Matlab®

Si se desean incluir procesos de procesamiento o análisis de selañes EEG desarrollados o implementados en Matlab® se debe tener instalado el runtime de MATLAB®, el cual se instala ejecutando el siguiente programa:

C:\Program Files\MATLAB\R2009a\toolbox\compiler\deploy\win32\MCRInstaller.exe.
(No es indispensable para el funcionamiento del VISOR EEG_M).

El proceso de construcción de estos módulos DLL están descritos en Modulos.hlp

MATLAB®, es un lenguaje de programación amigable al usuario con características avanzadas y mucho más fáciles de usar que otros lenguajes de programación como BASIC o C, además de ser un programa con un gran desempeño para el cálculo numérico, el procesamiento de señales e imágenes y la visualización gráfica. Cuenta con una gran cantidad de *Toolboxes* en el campo del procesamiento y tratamiento de señales e imágenes.

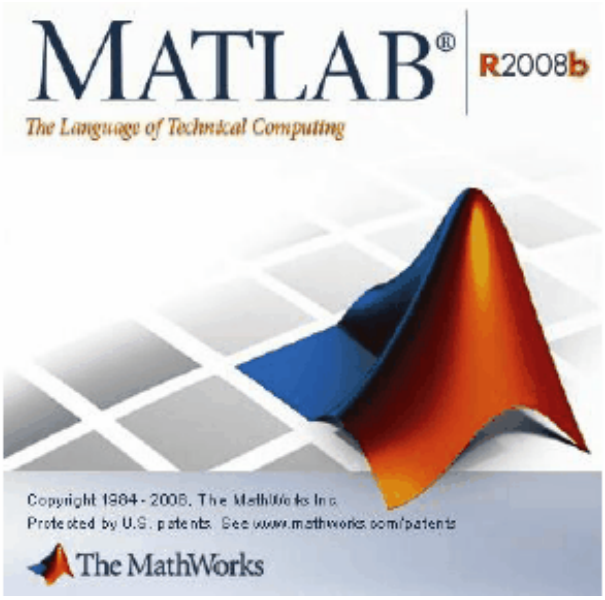


Figura 1.3 Interfaz de Matlab®

C++

Si se desean incluir procesos de procesamiento o análisis de señales EEG desarrollados o implementados en C++, estos archivos DLL se deben incluir en la carpeta módulos C:\Archivos de Programas\VisorEEG_M) . (C++ no es indispensable para el funcionamiento del VISOR EEG_M).

El proceso de construcción de estos módulos DLL están descritos en Modulos.hlp

INSTALACIÓN DE MÓDULOS

El visor EEG_M permite incluir módulos adicionales elaborados en lenguajes que generen código ejecutables como C, C++, Pascal, etc. o lenguajes interpretados como Matlab®, Java, etc.

Un módulo consiste en una sección de programa creada y/o compilada de forma independiente del VisorEEG_M, pero que puede incorporarse a éste, si cumple con los prototipos que se detallan en modulos.hlp.

Una vez construidos los módulos, se deben copiar en la carpeta módulos (C:\Program Files\VisorEEG_M\Modulos) cuando son archivos DLL's. Si son otro tipo de archivos con instrucciones de un lenguaje interpretado (Matlab®, Java, etc.), se debe crear una subcarpeta con el nombre del lenguaje e incluirla en la carpeta módulos mencionada anteriormente (cada vez que se agregue un módulo, se debe crear una subcarpeta nueva, con el nombre del módulo). En esta subcarpeta se incluyen los archivos de funciones o procedimientos, al igual que el archivo soporte.DLL, que es el encargado de comunicarse con el Visor EEG_M, para poder ejecutar dichos archivos.

Nota: En el Visor EEG_M se incluye dos ejemplos simples; un módulo en versión DLL (*PrModulosSample1.dll*) con código fuente disponible en lenguaje C++, y otro módulo desarrollado en Matlab®, cuyos archivo se incluyen en la carpeta

C:\Program Files\VisorEEG_M\Modulos\matlab\statistics

ANEXO 4. GUÍA DE USUARIO DEL VISOR EEG_M

VISOR EEG_M

Visor EEG_M es una herramienta de software para la visualización y análisis de registros EEG, que permite la utilización de módulos independientes agregándole nuevas funciones para el procesamiento y análisis de las señales EEG. Estos módulos pueden ser creados utilizando diferentes herramientas de diseño de programas (C, C++ Builder, Matlab, etc.), siempre y cuando cumplan con unas especificaciones de diseño preestablecidas por los autores. Esta herramienta le permite al especialista en neurología o ciencias afines aplicar los avances en los algoritmos de procesamiento de señales elaborados por desarrolladores de software, quienes se ven beneficiados al tener una interfaz amigable ya desarrollada, que les permite concentrarse solamente en la implementación propia de los algoritmos o la aplicación de las técnicas de procesamiento de señales.

El Visor EEG_M tiene una interfaz que permite la visualización de las señales en diferentes escalas de tiempo/voltaje y la representación topográfica en 2D (mapeo 2D) de los registros. La visualización de los registros puede configurarse utilizando diferentes montajes preestablecidos o nuevos montajes creados por el usuario. Para todas las señales o para cada una en particular se pueden aplicar filtros digitales (pasa bajos, pasa banda o rechazabanda) configurables. También permite opcionalmente observar el video asociado al registro y las marcaciones incluidas en el momento del examen o hechas posteriormente por el especialista o por módulos adicionales que se le pueden incorporar al programa.

Visualmente está compuesto por 7 paneles que se pueden activar por separado:

1. 1D
2. 2D
3. CONFIGURACIÓN
4. VIDEO
5. EVENTOS
6. CONTROLES
7. LINEA DE TIEMPO

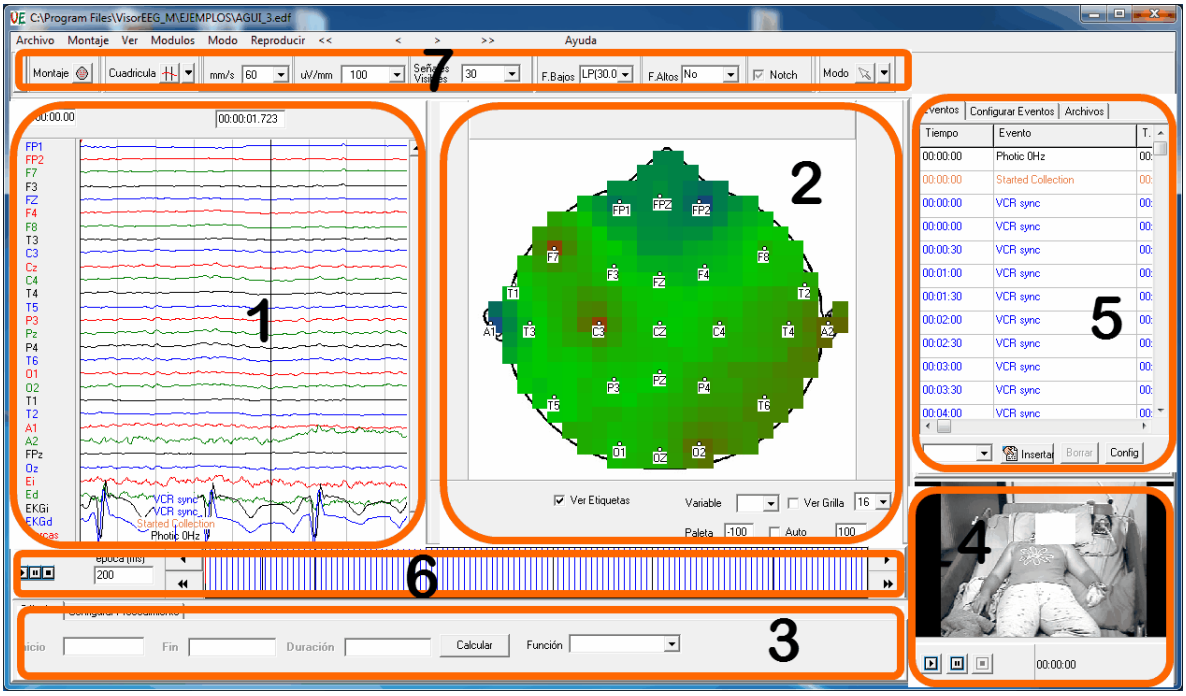
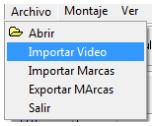


Figura 2.1 Interfaz de usuario del VISOR EEG_ M

MENÚ PRINCIPAL

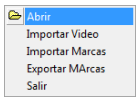
Figura 2.2 Menú Principal

Archivo



Permite acceder a las funciones normales que se ejecutan sobre un archivo que funciona en un ambiente Windows.

Abrir



Permite seleccionar el archivo de registros de EEG. Si existe un archivo de video y de marcas asociado a éste, también serán abiertos.

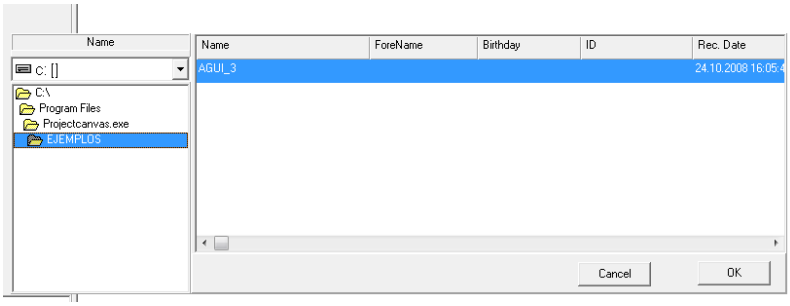


Figura 2.3 Ventana al ejecutar la opción de abrir

Importar Video

Permite abrir un video y asociarlo al archivo de registro (esto sucede cuando el archivo de registro no tiene el mismo nombre del archivo de video).

Importar marcas

Permite abrir un archivo de marcas y asociarlo al archivo de registro (esto sucede cuando el archivo de registro no tiene el mismo nombre del archivo de marcas).

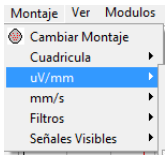
Exportar marcas

Permite generar un archivo ASCII correspondiente a las marcas hechas sobre el registro.

Salir

Permite salir correctamente de Visor EEG-M (opción más recomendada)

Montaje

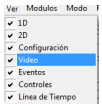


Permite configurar los diferentes parámetros de un montaje, así como también sus características de visualización. Las opciones disponibles son:

- Cambiar montaje
- Cuadrícula
- µV/mm
- Filtros
- Señales visibles

Esta información se amplía en la sección de PANEL DE CONTROLES

Ver



Activa o desactiva la visualización de los diferentes paneles

1D
CONFIGURACION
EVENTOS
LINEA DE TIEMPO

2D
VIDEO
CONTROLES

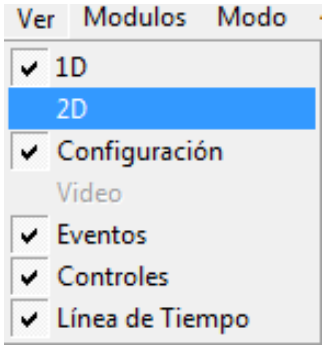
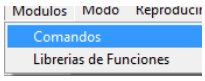


Figura 2.4 Pestaña ver

Mayor información en la sección paneles

Módulos



Permite verificar los módulos instalados en el programa, listando los procedimientos, funciones, tipos de datos y controles declarados en cada uno de los módulos, y también permite utilizar ventanas de comandos propias de los lenguajes interpretados que estén instalados (Matlab).

Comandos

Muestra los intérpretes de comandos instalados en el Visor EEG_M, que permite hacer pruebas rápidas de aplicaciones hechas en algún lenguaje interpretado del cual se tenga el soporte (previamente instalado).

MatLab

Muestra el interprete de comandos de Matlab, para ejecutar funciones, ver historial, ver variable, etc. Simula el entorno de Matlab para hacer ciertos ensayos; se debe tener instalado el *Engine* de Matlab.

Módulos instalados

Está opción queda habilitada para mostrar los módulos propios de análisis o procesamiento de señales EEG que se adicionen al Visor EEG_M (el cómo se construyen se indica en el archivo modulos.hlp).

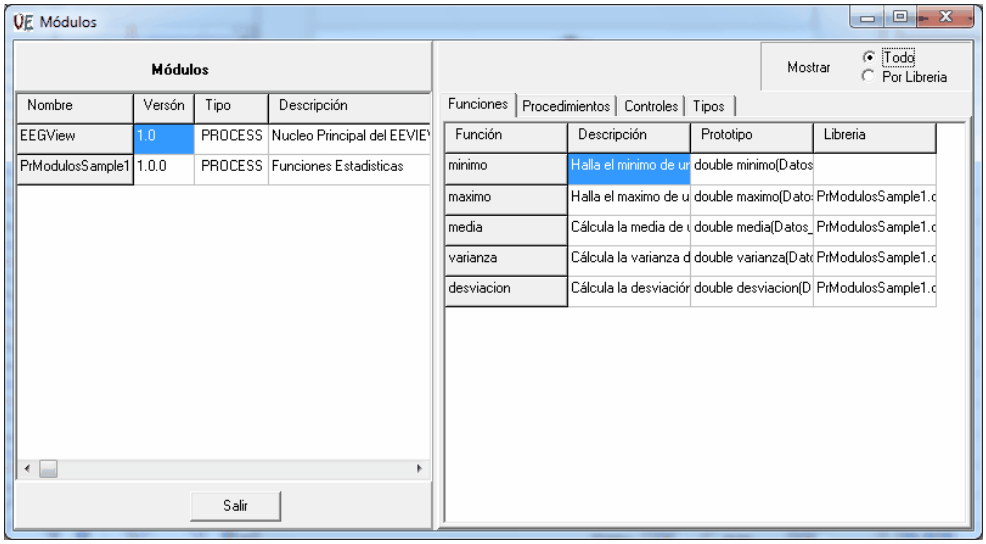
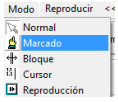


Figura 2.5 Ventana que ilustra los módulos instalados y sus funciones

Al hacer click en esta opción se ilustra un cuadro de diálogo que muestra una breve descripción de las funciones, procedimientos, controles y tipos de datos de todos o cada uno de los módulos disponibles.

Modos



Indica la forma de funcionamiento del cursor sobre la ventana de visualización del panel 1D

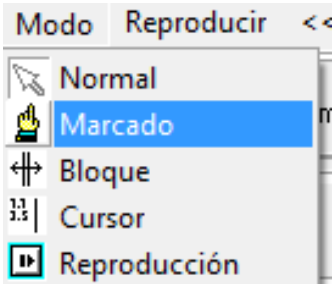


Figura 2.6 Pestaña Modos

- **Normal**
- **Marcado**
- **Bloque**
- **Curso**
- **Reproducción**

Normal

En este modo el usuario puede manipular los controles de montaje (escalas, filtros, etc.), avanzar retroceder en la línea de tiempo, pero no puede insertar o borrar marcas.

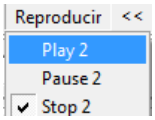
Marcado
Este modo se utiliza para modificar las marcas que contiene un archivo, agregar, quitar o configurar marcas.

Bloque
Selecciona una porción del registro para aplicar algunas de las funciones del procedimiento

cursor
Permite mostrar alguna función para un instante de tiempo para una señal elegida por el usuario.

Reproducción
Permite ver el video en modo continuo o por instantes discretos de tiempo predefinidos por el usuario.

REPRODUCCIÓN REGISTROS



Herramienta para avanzar, pausar o detener automáticamente en el tiempo la visualización del registro en 1D o 2D. Para el panel de video este proceso hace que se visualice el *frame* más cercano al instante de tiempo.

Avance
Herramienta para avanzar automáticamente en el tiempo la visualización del registro (el periodo del avance se indica en el control época del panel línea de tiempos).

Pausa
Herramienta para pausar automáticamente en el tiempo la visualización del registro.

Detener
Herramienta para detener automáticamente en el tiempo la visualización del registro.

DESPLAZAMIENTOS

Son utilizados para el desplazamiento del registro activo

Figura 2..7 Pestaña para desplazamientos del registro

<<

Permite el desplazamiento a la izquierda (hacia atrás) por página en el tiempo del registro EEG.

>

Permite el desplazamiento a la derecha (hacia adelante) por segundo (o una unidad de la cuadrícula gruesa) en el tiempo del registro EEG.

<

< Permite el desplazamiento a la izquierda (hacia atrás) por segundo (o una unidad de la cuadrícula gruesa) en el tiempo del registro EEG.

>>

>> Permite el desplazamiento a la derecha (hacia adelante) por página en el tiempo del registro EEG.

Ayuda

Contiene el manual de usuario y las ayudas requeridas para el manejo del Visor EEG_M

PANELES

EEG - M VISOR es una plataforma que nos permite visualizar registros EEG y hacer el mapeo en 2D, visualmente está compuesto por 7 paneles los cuales se pueden activar por separado:

1. 1D
2. 2D
3. CONFIGURACIÓN
4. VIDEO
5. EVENTOS
6. CONTROLES
7. LINEA DE TIEMPO

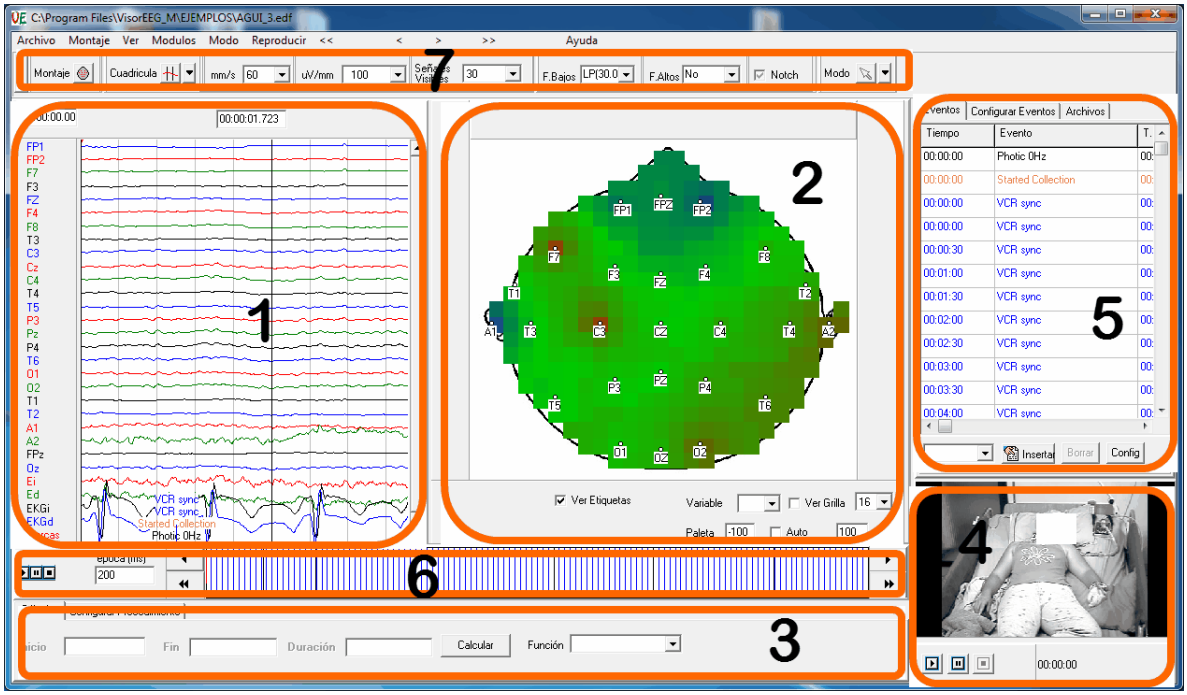


Figura 2.8 Paneles del Visor EEG_M

PANEL 1D

Panel principal donde se visualiza las señales que componen el registro y que son configurables según el montaje.

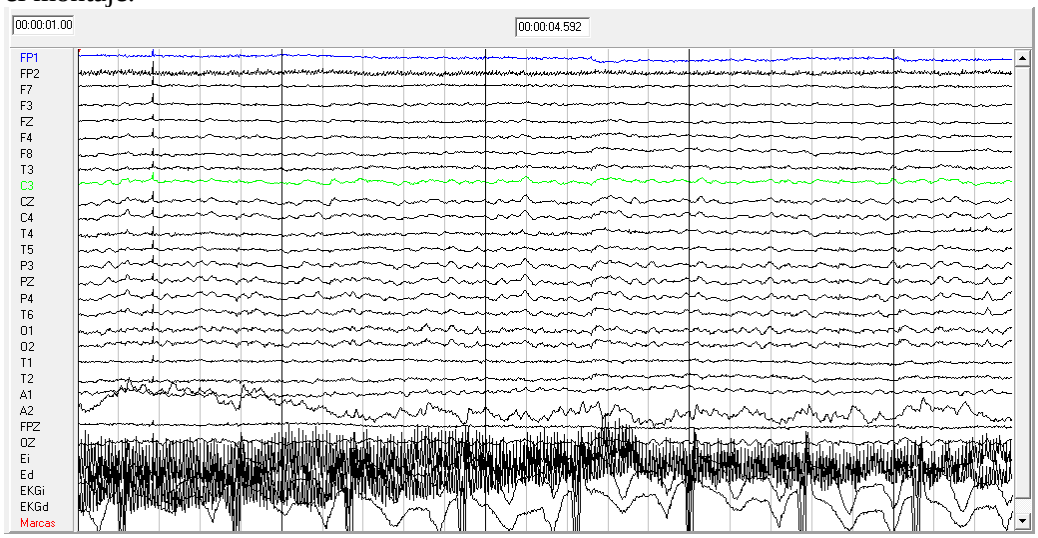


Figura 2.9 Panel 1D

NOTA: el video, el registro EEG y el mapeo 2D siempre estarán sincronizados.

Ventana de visualización de registro EEG

En la cual se observa todas la gráficas *v vs t* de cada una de las derivaciones y las marcas asociadas al registro, puestas por el especialista o por algún proceso del visor. Los colores y las amplitudes son configurables y dependen del montaje o de la configuración del usuario. La cuadrícula se puede modificar según necesidad del usuario (Simple – doble - ninguna).

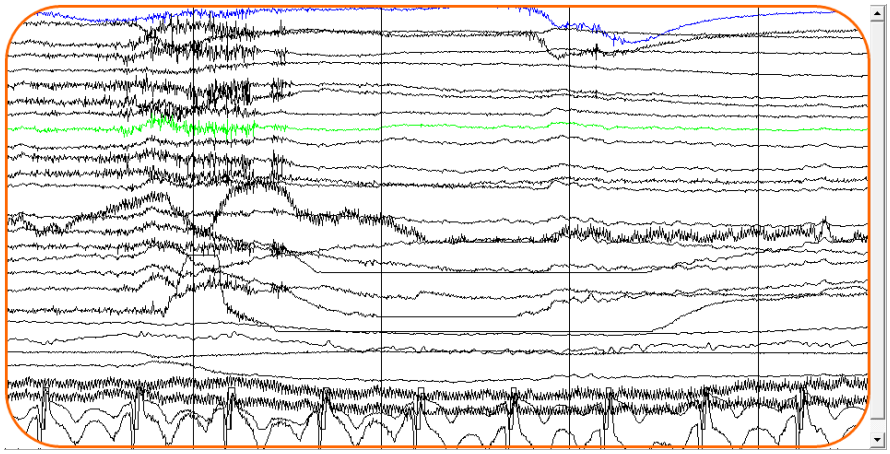


Figura 2.10 Ventana de visualización

Tiempo inicial

Indica el tiempo en el que empieza la ventana activa del registro que se observa en pantalla, es modificado cada vez que se desplaza por el registro.

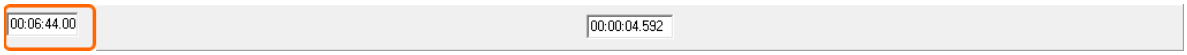


Figura 2.11 Ventana tiempo inicial

Rango de Tiempo

Indica el rango de tiempo del registro que se visualiza en la ventana activa. Es modificado según el factor de la escala de tiempo (mm/s).

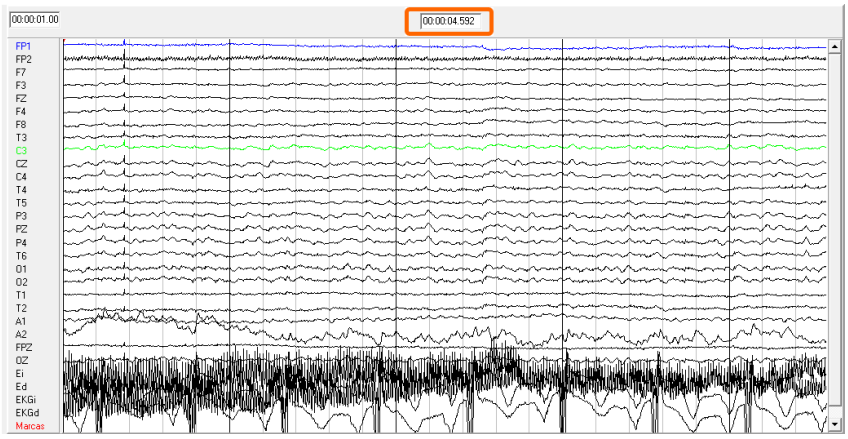


Figura 2.12 Ventana rango tiempo

Subpanel de derivaciones

Permite visualizar el nombre de cada una de las derivaciones del registro en pantalla. Varía según el montaje seleccionado. Es posible seleccionar una o varias señales para posteriormente aplicarle alguna técnica de procesamiento o configurar las características de visualización, una vez se le aplique algún procesamiento a uno o varios canales es posible que se modifique la información indicando el resultado del proceso.

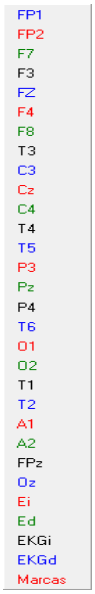


Figura 2.13 Subpanel derivaciones

PANEL 2D

Panel que permite ver un mapeo topográfico del registro EEG

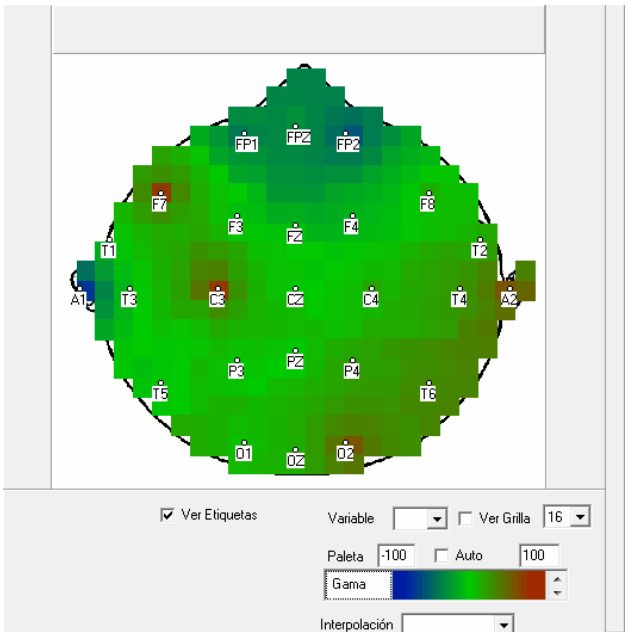


Figura 2.14 Panel 2D

NOTA: el video, el registro EEG y el mapeo 2D siempre estarán sincronizados.

Barra de desplazamiento vertical

Si el usuario configura la ventana de visualización de registro EEG de tal manera que el número de canales (derivaciones) sea menor a la cantidad que contenga el registro, esté cursor permite desplegar diferentes canales en la ventana activa.

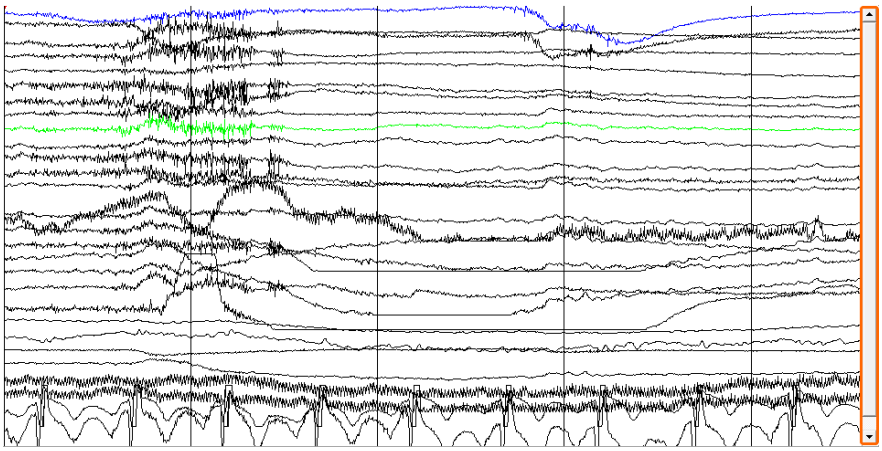


Figura 2.15 Barra de desplazamiento

Ver etiquetas

Habilita el nombre o etiquetas de las derivaciones

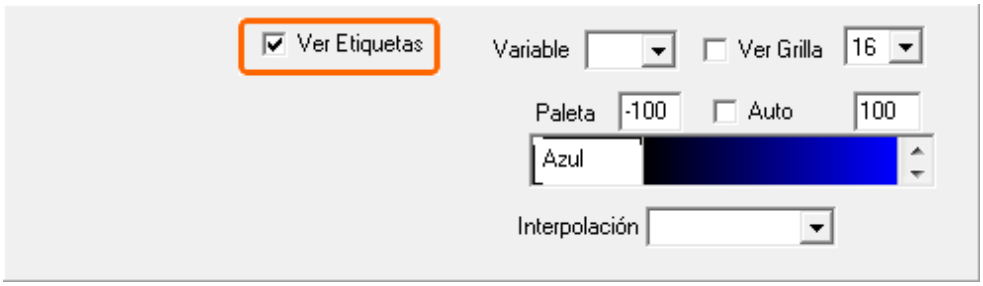


Figura 2.15.a Etiquetas

Variable

Elige la variable física que se desea visualizar.

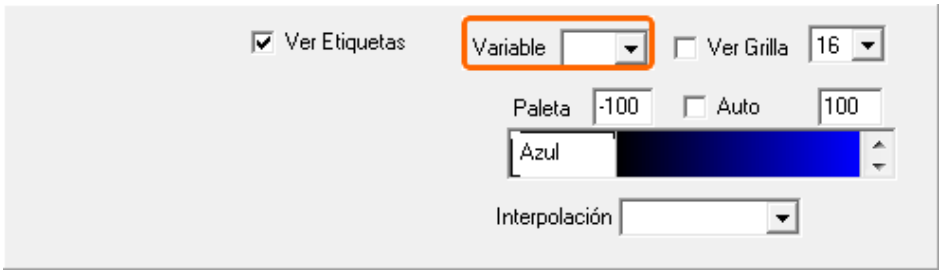


Figura 2.15.b Variables

Ver

Activa-desactiva la cuadrícula.

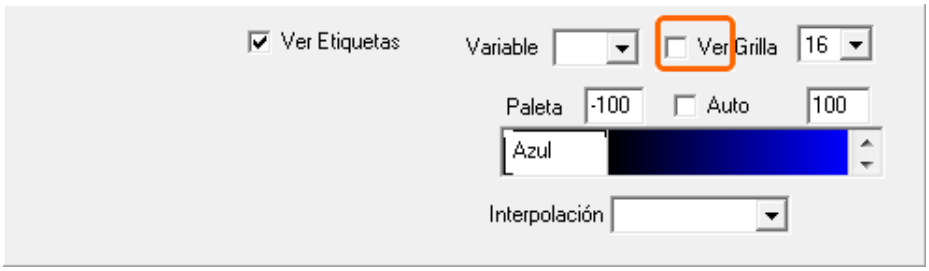


Figura 2.15.c Ver

Grilla

Define el tamaño de la cuadrícula.

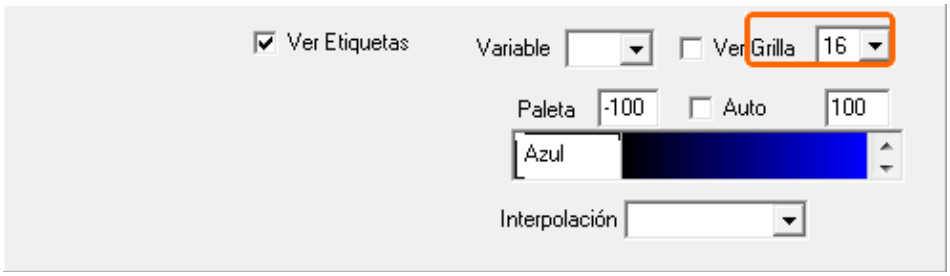


Figura 2.15.d Grilla

Paleta

Opción para cambiar los colores de visualización.

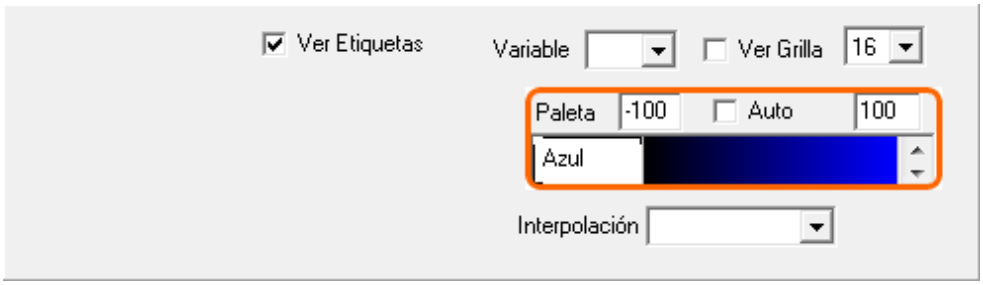


Figura 2.15.e Paleta

Auto

Activa – desactiva la opción para manejar manualmente o automáticamente los máximos y mínimos en el escalado de los colores.

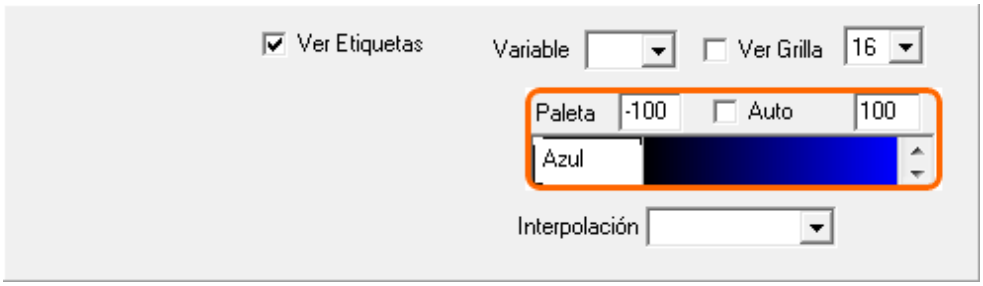


Figura 2.15.f Auto

Interpolación

Selecciona los diversos algoritmos para la interpolación.

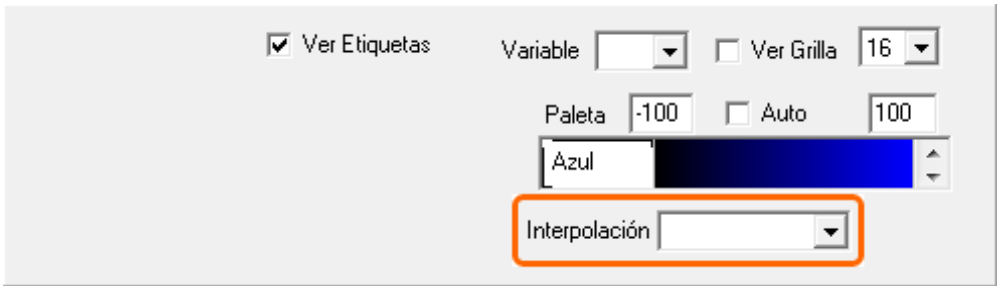


Figura 2.15.h Interpolación

PANEL CONFIGURACIÓN

Panel que permite calcular algunos parámetros estadísticos propios de las señales (máximos, mínimos, media, varianza, desviación, etc.) y configurar los diversos procedimientos desarrollados. De igual manera, se pretende que por medio de este panel se puedan calcular otros parámetros o funciones, implementadas en una DLL o un *script* de *Matlab*.

NOTA 1: Estas funciones son activas SOLO en modo bloque
NOTA 2: Recuerde seleccionar previamente la Señal (derivación) y marcar el bloque

Figura 2.16 Panel Configuración

Pestaña Cálculo

Ventana que muestra los parámetros (inicio, fin y duración o tamaño del bloque sobre el cual actúa la función) y resultados de las funciones invocadas en algunos procedimientos.

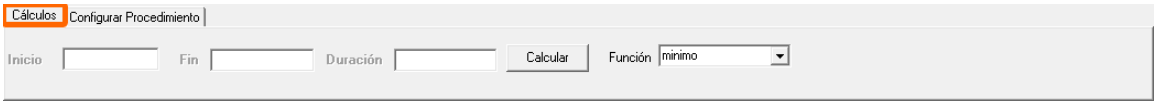
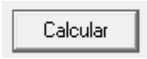


Figura 2.17 Pestaña cálculo

Calcular



Invoca la función indicada, los resultados son observados en el subpanel de derivaciones

Función

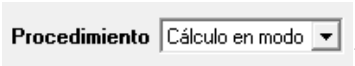
Permite seleccionar la función a ejecutarse.

Pestaña configurar procedimiento

Configura los diversos procedimientos de procesamientos de las señales EEG: el rango de operación de los mismos, los parámetros de análisis y la función a utilizar.

Figura 2.18 Pestaña cálculo

Procedimientos



Indica el procedimiento a configurar (selección o página completa, u otro instalado posteriormente).

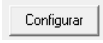
Funciones

Permite seleccionar las funciones de análisis de señales asociadas al procedimiento en cuestión.

F. predeterminada

Indica la función por defecto que se utilizará en el procedimiento, la cual puede ser modificada.

Configurar



Activa el cuadro de dialogo del procedimiento en cuestión.

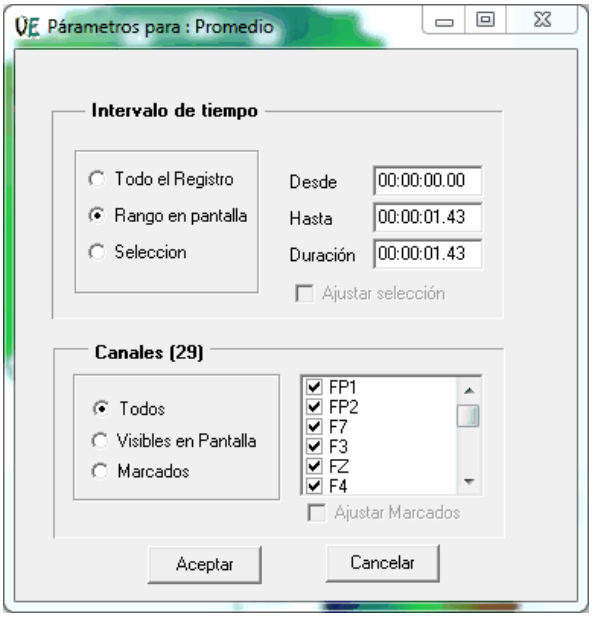


Figura 2.19 Ventana

PANEL VIDEO

Panel que muestra el video (si existe) del registro EEG, contiene los elementos básicos de reproducción (*play*, *pause* y *stop*) y el tiempo transcurrido.

Figura 2.20 Panel Video

NOTA: El video, el registro EEG y el mapeo 2D siempre estarán sincronizados.

Barra de tiempo

Permite posicionarse en un instante de tiempo específico del video.

Controles de reproducción

Contienen las herramientas básicas de reproducción de video: *Play*, *Pause* y *Stop*

PANEL EVENTOS

Visualiza y configura los diversos eventos o marcas etiquetados durante el examen, marcados por el especialista o generados automáticamente por algún procedimiento instalado en el programa.

Figura 2.21 Panel Eventos

Pestaña eventos

Muestra una lista de los eventos en forma ascendente en el tiempo (los tipos de eventos que aparecen en este listado se pueden ser activar o desactivar por medio de la pestaña de configuración de eventos), igualmente se puede ver el inicio y el fin de cada evento.

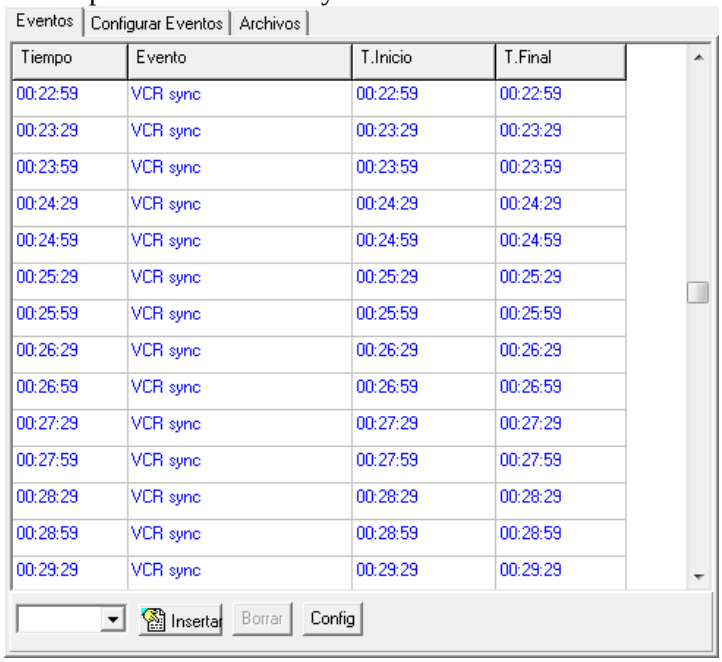


Figura 2.22 Pestaña Eventos

Pestaña de configuración de eventos

Activa, desactiva, crea y/o configura los tipos de eventos que aparecen en la pestaña de eventos y en la línea de tiempo, la configuración incluye el color, si es o no visible y la tecla rápida utilizada para marcar el evento.

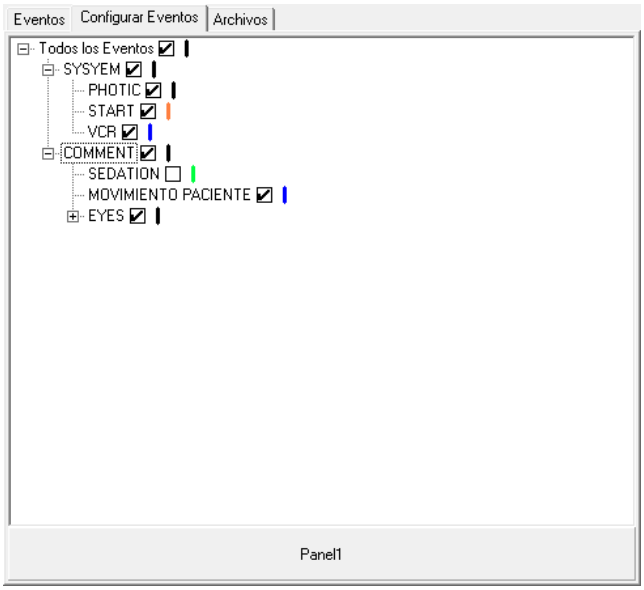


Figura 2.23 Pestaña configuración de eventos

Pestaña archivo

Permite activar o desactivar archivos de marcación en la visualización.

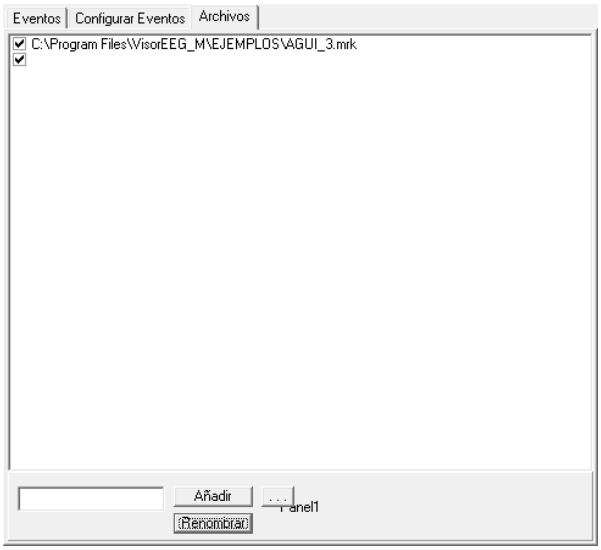


Figura 2.24 Pestaña archivo

PANEL CONTROLES

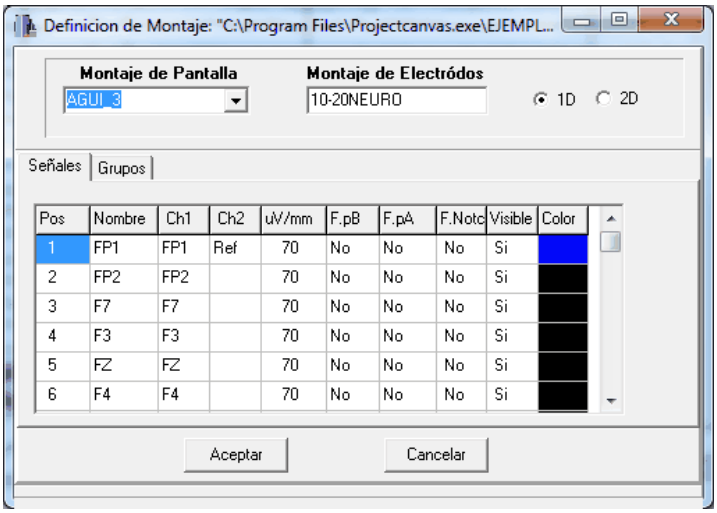
Permite la manipulación de los comandos más utilizados en la visualización de un registro. Cada vez que se instalan módulos que contienen controles, estos serán agregados a este panel. Para mayor información en la sección MONTAJE de la unidad uno.

Figura 2.25 Panel controles

Cambiar Montaje

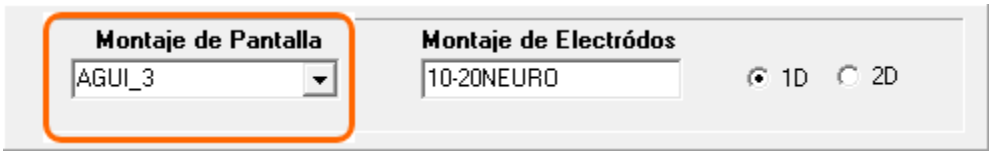
Permite definir nuevos montajes o modificar los existentes y asociárselos a los registros.

Figura 2.26 Ventana que permite configurar el montaje



Montaje de Pantalla

Selecciona el montaje de los ya existentes o permite definir uno nuevo.



TIPOS DE MONTAJES

- Bipolar
- Lateral
- Transversal
- Ref (Referenciado a tierra)
- Nuevo

Montaje de Electrodos_2

Información de la configuración de electrodos con que fue elaborado el EEG (no permite ser modificado)

1D- 2D

Da la opción de ver el montaje en una o dos dimensiones.

Señales

Permite modificar las propiedades de visualización de cada una de las señales (Nombre, Canales, Sensitividad, Filtros y Color).

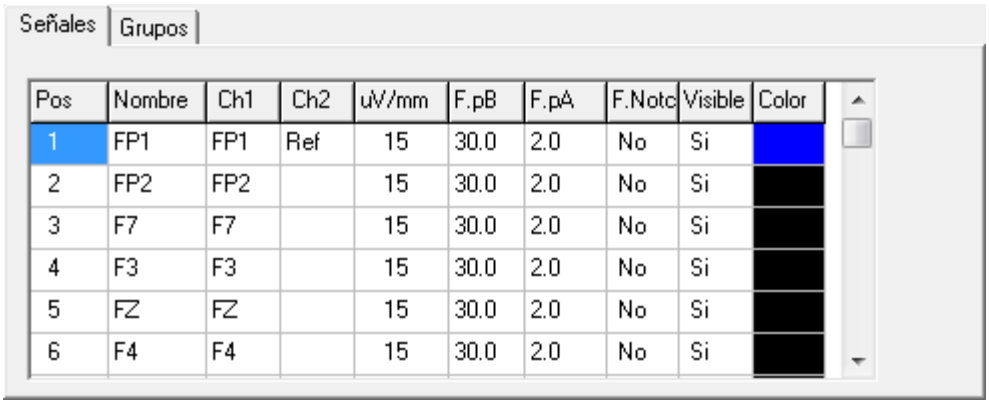


Figura 2.27 Pestaña SEÑALES que permite seleccionar y configurar las propiedades de visualización para el montaje en 1D

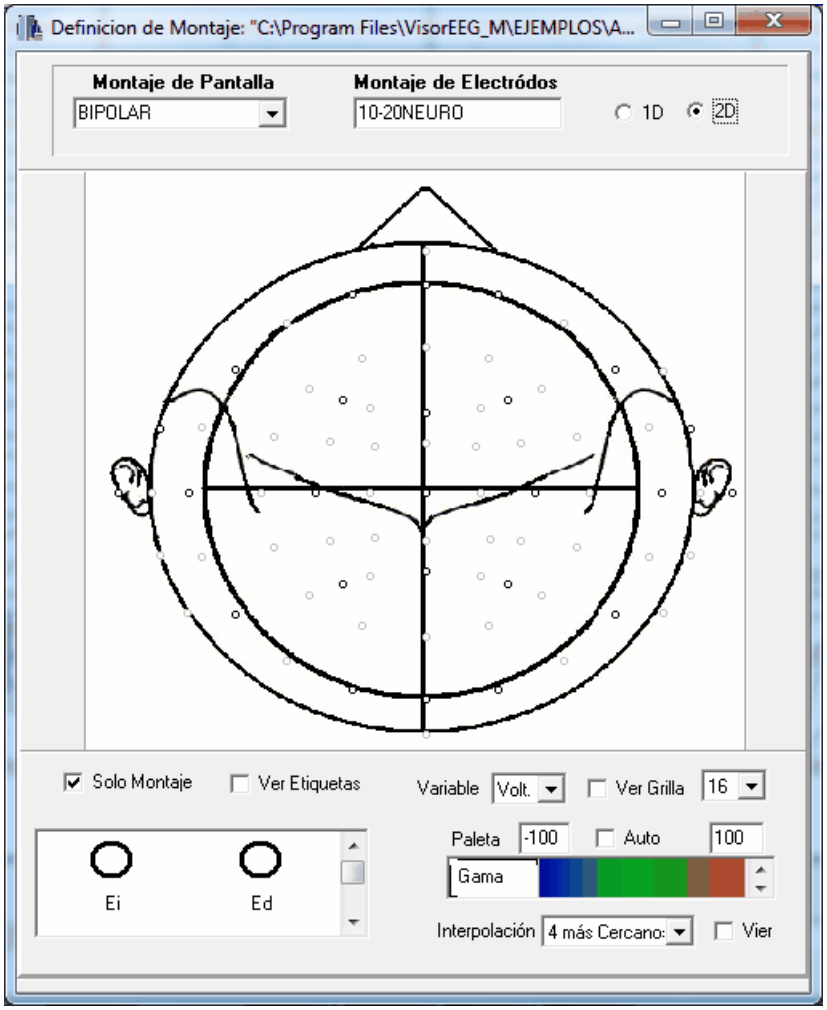
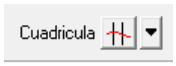


Figura 2.28 Pestaña SEÑALES que permite seleccionar y configurar las propiedades de visualización para el montaje en 1D

Cuadrícula_2



Establece una cuadrícula al registro para facilitar la observación del mismo (Simple – Doble – Ninguna)

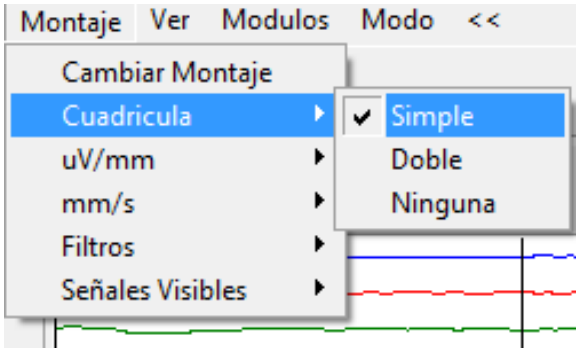
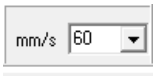


Figura 2.29 Pestaña cuadrícula

mm/s (Escala en el tiempo)_2



Permite variar la escala del tiempo en una escala de 0.5 – 60 mm/s (eje horizontal), se puede actuar sobre todas las señales, o por separado, seleccionando previamente la(s) señal(es) que se desee modificar.

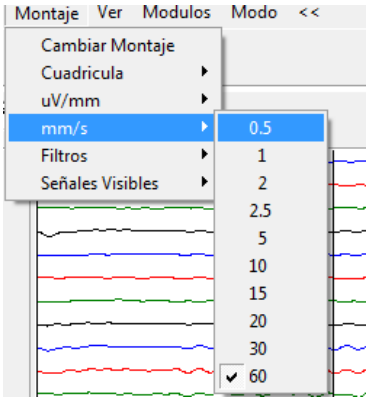
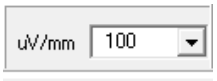


Figura 2.30 Pestaña escala en el tiempo (mm/s)

NOTA: Cuando en este cuadro no se indica ningún valor, significa que existen señales con diferente escala de tiempo.

uV/mm (sensibilidad)_2



Permite variar la amplitud de las señales en una escala de 1000 -1 uV/mm (eje vertical), se puede actuar sobre todas las señales, o por separado, seleccionando previamente la(s) señal(es) que se desee modificar.

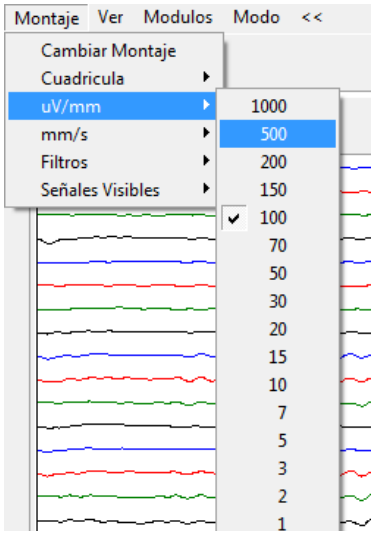


Figura 2.31 Pestaña sensibilidad (uV/mm)

NOTA: Cuando en este cuadro no se indica ningún valor, significa que existen señales con diferente escala de amplitud.

Señales visibles_2

Permite indicar el número de canales visibles simultáneamente en pantalla, con la barra de desplazamiento vertical se pueden ir observando el registro completo.

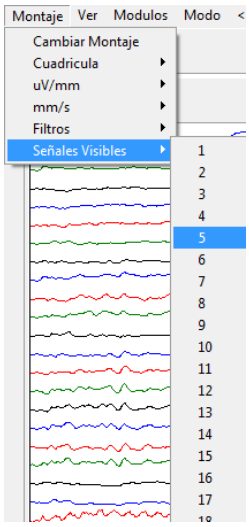
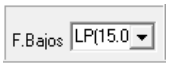


Figura 2.32 Pestaña señales visibles

Filtros_2



Permite aplicar un filtro a todas las señales o a una en particular (Pasa bajos, Pasa altos, rechazabanda), al seleccionar el Filtro pasabajos o el pasa altos se puede indicar la frecuencia de corte 15-100Hz y 0.5 -15 Hz respectivamente, en el *Notch* la frecuencia de rechazo es de 60 Hz por defecto.

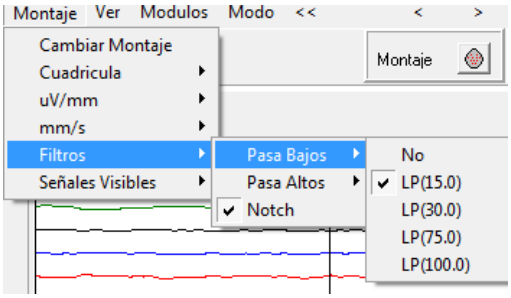


Figura 2.33 Pestaña filtros

PANEL LÍNEA DE TIEMPO

Permite el desplazamiento en el tiempo del registro e indica las marcas con líneas verticales.

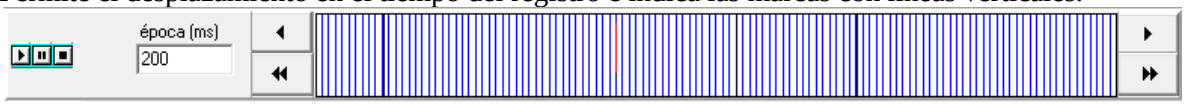
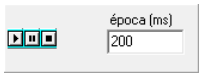


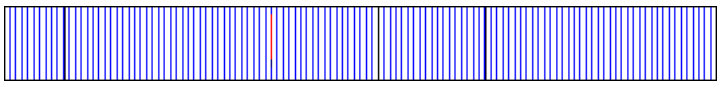
Figura 2.25 Panel línea de tiempo

Controles



Utilizados para el desplazamiento del registro activo

Línea eventos


Indica la posición de las diversas marcas de los eventos durante un registro de un EEG.

ANEXO 5. CD DE INSTALACIÓN DEL VISOR EEG_M (INCLUYE AYUDAS)